SYNTHETIC SKIN FOR YOUR ROBOT AND HOW TO MAKE IT

THE CODE THAT CAN'T BE CRACKED

TLC—THE VISUAL PROGRAMMING LANGUAGE

MICROCOMPUTERS HELP THE DEAF-BLIND

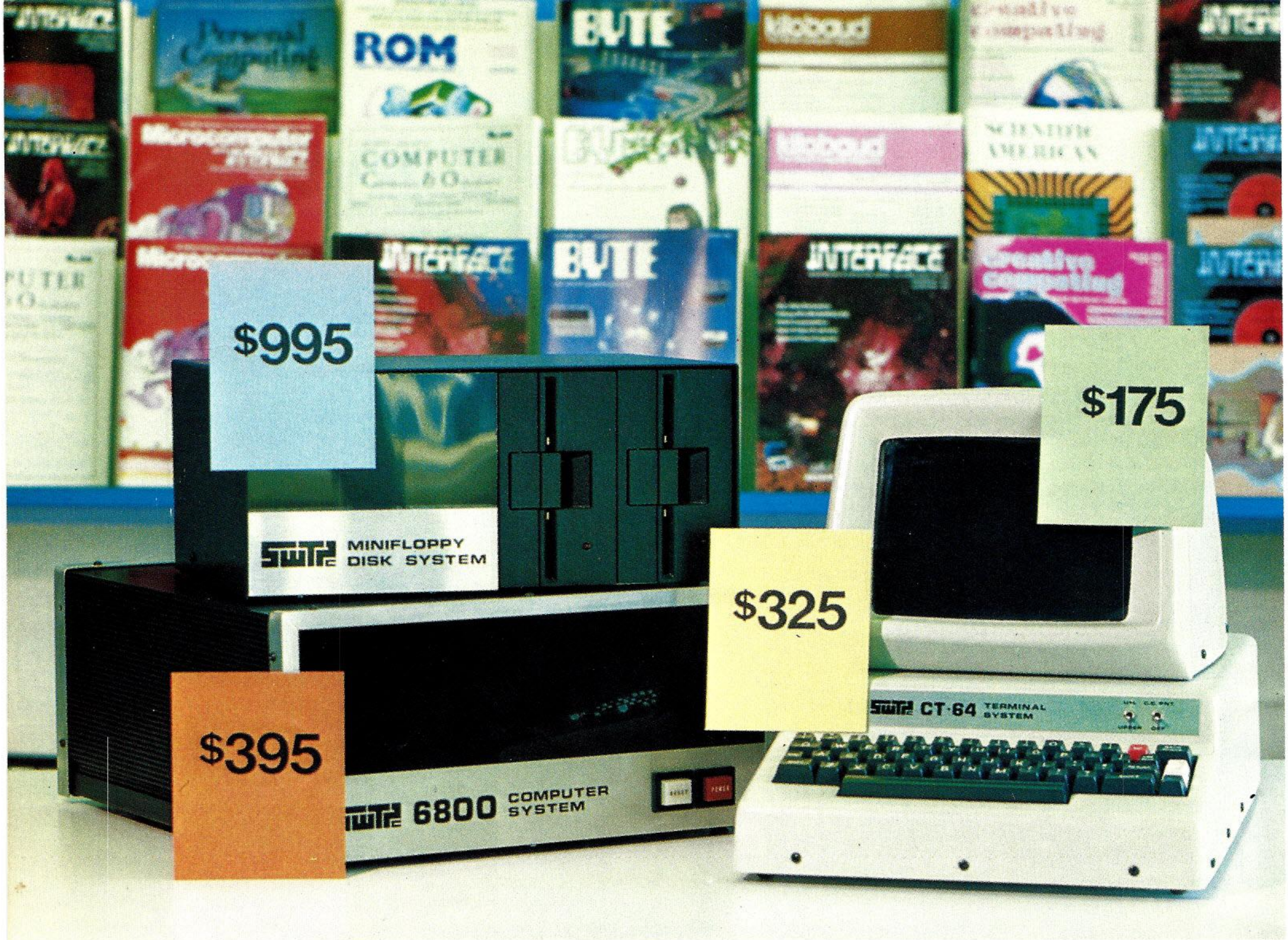
ROM

COMPUTER APPLICATIONS FOR LIVING

VIDEO-SKETCH
Micro Graphics Program

Volume I, Number 7
January 1978/$2.00

8080
6800
S100
G.HAIN


$995
SWTPC MINIFLOPPY DISK SYSTEM
$175
$325
SWTPC CT-64 TERMINAL SYSTEM
$395
SWTPC 6800 COMPUTER SYSTEM

Your computer system needn't cost a fortune.

Some computer kits include little more than a mother board and a front panel, and you pay extra for everything else you need to make an operating computer.

SWTPC doesn't do it that way, so you can get your Southwest Technical 6800 Computer up and running at a bargain cost compared with most other systems. It comes complete at $395 with features that cost you extra with many other systems.

The Extras You Get

These extras include 4K of random-access memory, a mini-operating system in read-only memory, and a serial control interface. They give you 1) a considerable amount of working memory for your programs, 2) capability through the mini-operating system to simply turn on power and enter programs without having to first load in a bootstrap loader, and 3) an interface for connecting a terminal and beginning to talk with your computer immediately.

Low-Cost Add-Ons

Now that you have a working computer, you'll probably want to add at least two features soon, more memory and interfaces for needed accessory equipment. Memory for our 6800 is another bargain. You can get 4K memory boards for just $100 and 8K boards for only $250.

Our interfaces cost little compared with many other systems. For just $35 you can add either a serial or parallel interface board. (And you won't have to buy several interfaces on a costly board to get just the one you want.)

Peripheral Bargains

Your computer is no good without at least a terminal for entering data and viewing computer output, and you will probably want a good method of storing programs and data.

We offer you a line of high-quality peripherals at low prices. (You can prove this by just comparing prices.)

Buy our CT-64 Video Terminal for only $325 and our CT-VM Monitor with matching cover for $175. Our MF-68 Dual Minifloppy costs just $995, complete with Disk BASIC and a disk operating system. For cassette storage our AC-30 Cassette Interface gives simple control for one or two cassette recorders.

You can get inexpensive hard copy with our PR-40 Alphanumeric Line Printer.

We back up the 6800 system with low-cost software, including 4K and 8K BASIC.

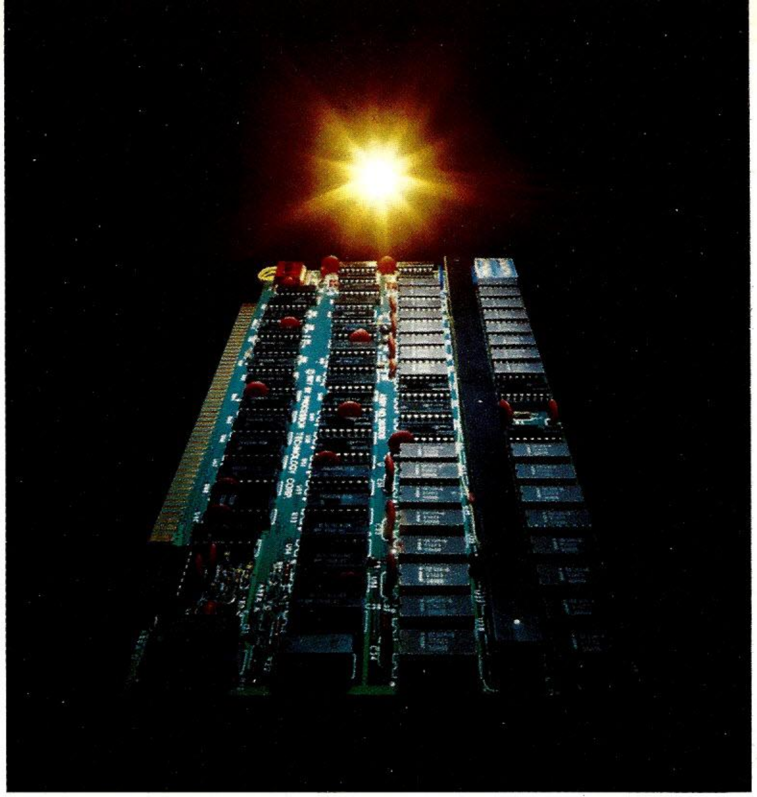
Compare the value you get with our computer and peripherals before you buy. We think you'll find that SWTPC gives you more for your money in every way.

Enclosed is:

______ $995 for the Dual Minifloppy
______ $325 for the CT-64 Terminal
______ $175 for the CT-VM Monitor
______ $395 for the 4K 6800 Computer
______ $250 for the PR-40 Line Printer
______ $79.50 for AC-30 Cassette Inferface
______ Or BAC # ____________ Exp. Date ______
______ Or MC # ____________ Exp. Date ______

Name ______________________ Address ______________________
City ______________________ State ______________ Zip ______

SWTPC Southwest Technical Products Corp.
219 W. Rhapsody, San Antonio, Texas 78216
London: Southwest Technical Products Co., Ltd.
Tokyo: Southwest Technical Products Corp./Japan



The end of Kit-Kits.

The end of bad solder joints, heat damaged components and sick IC's. Introducing the Semikit. Item 1, a 16KRA Memory Board, $369.

Let's face it. Loading and soldering PC Boards is not much fun for the kit builder. Even more important, it's the place where most of the trouble gets introduced. The real fun and education comes in running and testing boards.

Now the Semikit with fully tested IC's.

At the price of a kit, Processor Technology Corporation introduces the Semikit. It's a fully stuffed, assembled and wave soldered PC Board loaded with IC's that have gone through Q.C. and final check-out (a first in the industry).

We leave you the fun of testing with our fully documented set of instructions. We do the production tasks of loading, wave soldering and inspecting the boards. You do the more interesting and time consuming chore of testing and burning-in the boards.

The result is one sweet deal for both of us. You get a board where the primary causes of damage (poor solder joints, excess solder and bad IC's) are virtually eliminated. You get a board of highest professional quality. And we get the business!

The 16KRA Memory Board's at your dealer now.

Your Processor Technology dealer has the first Semikit, a 16KRA Memory Board, in stock and ready to go right now. You can take it home tonight for $369 as a Semikit or for $399 fully assembled, tested and burned-in.

You'll have a 16,384 byte memory with a better price performance ratio than anything on the market today. Now you can afford to add quality, high density memory to your system for remarkably little. And you can add enough to solve complex computing problems right in the main frame.

The memory features invisible refresh. There's no waiting while the CPU is running. Worst case access time is 400 nsec. Each 4,096 word block is independently addressable for maximum system flexibility. Power is typically 5 watts, the same as most single 4K memory modules. Back-up power connection is built-in.

Other Semi's are coming your way.

The 16KRA Memory is Processor's first step in adding more fun, capability and reliability to your computer system at lower cost. Other modules are on the way to your dealer now. Come on down today.

Or you may contact us directly. Please address Processor Technology Corporation, Box O, 7100 Johnson Industrial Drive, Pleasanton, California 94566. Phone (415) 829-2600.

Processor Technology

SELECTRIC TERMINAL

- **14.8 cps print speed**
- **13 inch line length**
- **10 or 12 characters/inch**
- **6 or 8 lines/inch**
- **IBM 2740-1, 2 line control**
- **IBM 2740 keyboard/printer**

This deal can't be beat. Not only do you get a quality terminal for a fraction of the cost of anything else on the market, but you also get tyewriter quality. Can be interfaced to a mini- or microcomputer or used with IBM computers. These terminals are worth the price of a typewriter alone Our warranty is limited to replacing any defective parts for a period of 90 days following receipt of equipment. Interchangeable print element, of course. Includes attractive desk. Optional documentation package $25.00.

SUPER SALE

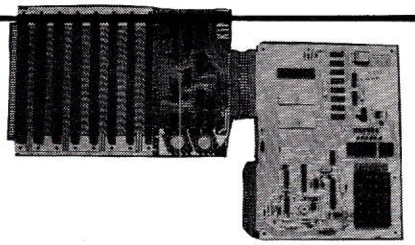
***JUST* $675**

COMPUTER DEALERS ASSOCIATION

CALL [313] 994-3200 TO RESERVE

KIM MEETS *S-100*

Item	Kit Price	Assembled Price
Kimsi Backplane/S-100 Adapter 8 slots for S-100 available (requires +8V, +16V & -16V unregulated power supply)	**$125** ☐	**$165** ☐
KIM to Kimsi Connector Set Includes recorder cables	**8** ☐	**11** ☐
KIM-1 incl. documentation Send for more information	**N/A** ☐	**245** ☐
S-100 Edge Connector Kimsi accepts 8; 1 is supplied	**5** ☐	**7** ☐
Cassette Recorder for KIM-1 Hundreds sold for KIM use	**N/A** ☐	**55** ☐
4K Imsai RAM Board Bare minimum to run Tiny BASIC	**139** ☐	**239** ☐
8K Seals RAM Board Enough space for programs below	**N/A** ☐	**269** ☐
Poly Video Terminal Interface 16 x 64 Connects to a Monitor & Keyboard	**210** ☐	**280** ☐
Matrox ALT-2562** Video Graphics Board displayed 256 x 256 array	**N/A** ☐	**395** ☐
Itty Bitty Tiny BASIC Uses 2.5K; kit a paper tape, assembled a cassette	**5** ☐	**10** ☐
Focal [a DEC trademark] Includes floating point	**N/A** ☐	**50** ☐
6502 Assembler & Editor This is NOT the MOS version	**N/A** ☐	**40** ☐



Escape the single source blues. The Kimsi is the easy way to expand your KIM-1 system with readily available S-100 boards. Forethought Products has interfaced the 6502 to the S-100 bus by decoding the top 4K for IN and OUT instructions. Full capacity interfacing allows complete control of the 6502 and DMA from the S-100 bus.

SPECIAL! 10% off S-100 Boards when ordered with Kimsi.

Name ______________________
Address ______________________
City State Zip ______________________
Charge my BAC/VISA____ M/C____ Interbank # ________
Card # ______________________ Exp. Date ________
Signature ______________________
☐ Please send me info on the new Commodore PET 2001 computer

Total for goods checked above
+4% Michigan Residents
+4% Shipping and Handling
+$1 if order's under $20
Total Amount

Limited 90-day Warranty
Return non-working merchandise within 90 days for replacement or refund. Kits are warranted to be complete with working components. All items subject to prior sale.

Newman Computer Exchange/CompuMart

1250 North Main Street Department **R1** P.O. Box 8610 Ann Arbor, Michigan 48107
(313)994-4445

Contents

Volume I, Number 7
January 1978

ROM
COMPUTER APPLICATIONS FOR LIVING

FEATURES

22 Synthetic Skin for Your Robot and How to Make It by Stephen J. Chiodo
Fleshing out your robot, be it a simple turtle or a complete android, is no problem if you use this film animator's construction techniques.

30 The Code That Can't Be Cracked by Eben Ostby
Actually, a computer could crack this code—given two or three million years.

36 TLC: The Visual Programming Language by Tony Karp
A new programming language that's almost as easy to write as English. Plus an introduction to its sibling, the "flowgram"—the easy symbolless way to chart programs.

54 A Beginner's Guide to Computer Graphics: or Raster Scan Can by Bill Etra
An introduction to videographic display, complete with a Video-Sketch program that turns your computer into a drawing board.

58 Home Computers: A Look at What's Coming by Jules H. Gilder
A quick rundown on the ever-changing hardware scheme.

64 The Computer and Natural Language by Joseph Weizenbaum
If you can't communicate with your computer, it's just a big old dumb machine. If you speak the same language. . . .

74 Microcomputers Help the Deaf-Blind by Frederick M. Kruger
Looking for a really useful microproject? Look no further.

87 First Timer's Guide to Circuit Board Etching by Thorn Veblen
For some projects there are no kits. You have to start from scratch. Here's how.

88 Call Exit (A short story) by Frederick W. Chesson
Old programmers never die, they just. . . .

DEPARTMENTS

4 On the Bus

6 Reader Interrupt

20 Eve 'n' Parity

48 Run On Micros

50 Centerfold

100 PROMpuzzle

COLUMNS

8 Missionary Position by Theodor Nelson
The Art of the Computer Screen

14 AIQuotient by A. I. Karshmer
Computer Vision

20 The Human Factor by Andrew Singer
Why Not Do It by Hand?

93 Legal ROMifications by Peter Feilbogen
Warranties?

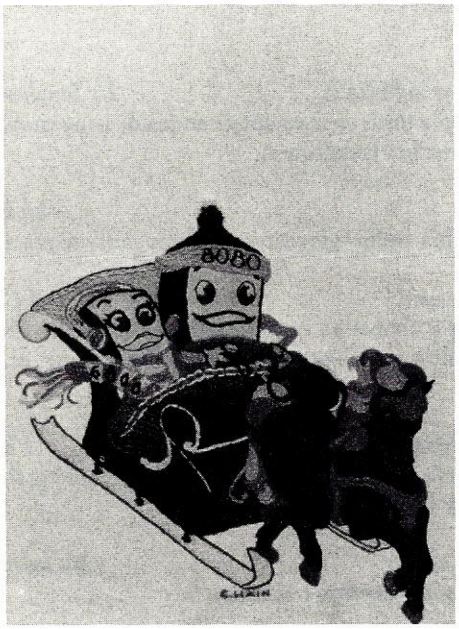
94 PROMqueries by Eben Ostby
Have a problem?
Ask ROM

96 FutuROMa by Bill Etra
The Capitalistic Computer

98 Cryptic Computer by Frederick W. Chesson
Computers and Cryptography

ROM is published monthly by ROM Publications Corporation, Route 97, Hampton, CT 06247 (Tel. 203-455-9591). Domestic subscriptions are $15 for one year, $28 for two years, and $39 for three years. Canada and Mexico $17 for one year, $30 for two years, and $41 for three years. For European and South American subscriptions, please add $12 per year additional postage. For all other continents, please add $24 per year additional postage. *Copyright © 1977 by ROM Publications Corporation. All rights reserved.* Reproduction in any form or by any means of any portion of this periodical without the consent of the publisher is strictly prohibited. The following trademarks are pending: AIQuotient, Babbage and Lovelace, Cryptic Computer, Eve 'n' Parity, floppyROM, futuROMa, The Human Factor, Legal ROMifications, Missionary Position, The Noisy Channel, On the Bus, PROMpuzzle, PROMqueries, Reader Interrupt, ROMdisk, ROMshelf, ROMtutorial, and Run On Micros. Opinions expressed by authors are not necessarily those of *ROM* magazine, its editors, staff, or employees. No warranties or guarantees explicit or implied are intended by publication. Application to mail at second-class postage rate pending at Hampton, CT 06247. Membership in Audit Bureau of Circulation pending.

On the Bus



Cindy Hain is a recent graduate of Parsons School of Design who has been a Contributing Artist at *ROM* since the beginning. She enjoys designing needlework as well as working with pen and ink or watercolor. Her favorite artists are Paul Giovanopoulos and Andrew Wyeth. What she wants most in the world now, she says, is to have Saturdays and Sundays off.

Hooked on crossword puzzles at an early age, **Daniel Alber** now constructs as well as solves them. Part of the brownstone renovation generation in New York, when he's not constructing puzzles for the likes of *Field and Stream, The New York Times,* and *ROM,* he's reconstructing olden golden rooms in his Brooklyn based house.

Frederick W. Chesson is a graduate of the University of Connecticut. After work in electronic engineering, he gravitated into technical writing. At present, he furnishes instruction manuals and related items to various firms plus construction articles to several electronics hobby magazines. A member of the American Cryptogram Association since 1958, he is currently researching a book on Civil War Codes and Ciphers.

Stephen J. Chiodo works as an animator, fabricator, cameraman, director, or producer—"whatever is needed at the time." For his film *Cricket* he was awarded the Best Young Director's Cup at the Cannes Film Festival. Chiodo says he believes in fantasy and adventure—"It is the motivating force in my life."

Bill Etra is a West Coast based computer design consultant. He is co-inventor of the Rutt/Etra Video Synthesizer—the first portable voltage control analog video synthesizer, as well as the Videolab. His main interest is videographics, and many of his works have appeared as cover illustrations on various periodicals and books including *Computers in Society* and *Broadcast Management and Engineering.* His current research centers on "The Computer as a Compositional Tool for Video."

Peter Feilbogen attended the Rutgers School of Business Administration and Brooklyn Law School. In addition to being an attorney, he is also a Certified Public Accountant. He has been a sole practitioner on Long Island for approximately ten years, and treasurer of Data Information Services, Inc. An avid tennis player, he is currently trying to improve his game with the aid of a computer.

Tony Karp runs TLC Systems, a New York consulting firm that specializes in the design of real-time systems for mini- and microcomputers. In the past he has worked as a writer, photographer, analog computer designer, and special-effects consultant for TV commercials. In 1970 he was nominated for an Academy Award for the design of a computer-controlled zoom lens used on *The Godfather.* His hobby is designing computer languages to be used by *people.*

A. I. Karshmer is currently completing his Ph.D. in computer science at the University of Massachusetts. His main

---

A Note to Contributors:
ROM is always looking for good computer applications articles from people with up-and-running systems. We also will be glad to consider for possible publication manuscripts, drawings, and photographs on other computer-related subjects. Manuscripts should be typewritten double-spaced, and a stamped self-addressed envelope of the appropriate size should accompany each unsolicited submission. Although we cannot assume responsibility for loss or damage, all material will be treated with care while in our hands. Contributions should be sent to ROM Publications Corporation, Rte. 97, Hampton, CT 06247.

interest is the use of artificial intelligence concepts in solving problems involved in the transmission of computer graphics. Currently, he is developing a method for sending high-density information, such as animated graphics, over existing low-bandwidth telecommunication networks.

Frederick M. Kruger has been Director of Research for the Helen Keller National Center for Deaf-Blind Youths and Adults since 1972. Before then, he was a research psychologist and taught neuropsychology at Queens College. Besides having received a Ph.D. in neuropsychology, Dr. Kruger has also had training in electronic engineering and computer sciences. Since 1974, he has been president of INTERCHANGE, the Interdata Computer User's Group.

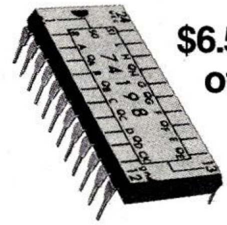
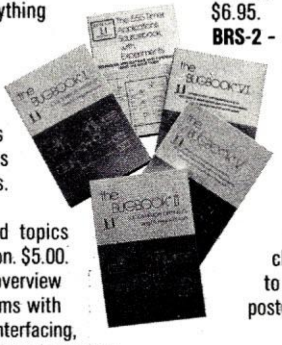
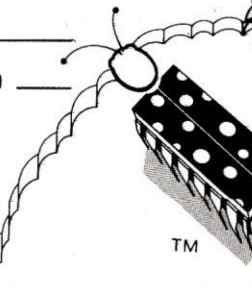
Theodor Nelson is the author of the classic *Computer Lib/Dream Machines,* a Whole Earth style catalogue of computer machinations. His latest book is the just-released *The Home Computer Revolution.* Ted specializes in highly interactive systems for graphics and text. His past experience includes a stint at Dr. Lilly's Dolphin Laboratory and work as a consultant for Bell Lab's ABM system.

Eben Ostby has been involved with computing ever since he crashed the PDP-8 at Pomfret School. At present, he is doing graduate work in computer science at Brown University and trying to convince people that APL isn't really all *that* bad.

Andrew Singer has been hooked on computers since he first built one in 1958. A hard/software consultant, he is fluent in thirty computer languages and knows more than enough about twenty species of machine. His work has included the first medical information retrieval system based on ordinary clinical records, and a large and intricate system for interactive selection of data from public opinion polls. He believes that most software is poorly designed and unspeakably rude, and his Ph.D. research is aimed at improving the architecture and human engineering of interactive systems.

Thorn Veblen is a free lance economist specializing in human value analysis in bionics. He claims to have no upper-class pretensions except for a preference for playing croquet on putting-green-smooth lawns.

Joseph Weizenbaum is Professor of Computer Science at the Massachusetts Institute of Technology. He is best known to his colleagues as the composer of SLIP, a list-processing computer language, and for ELIZA, a natural-language processing system. More recently, he has directed his attention to the impact of science and technology—and of the computer in particular—on society.

Free Bugbacks.

$6.50 worth of Bugback® labels free with $35.00 book purchase.

Our Bugback adhesive labels stick to their chips, immediately identifying logic functions and pin numbers. But even without the $6.50 Bugback freebie, our Bugbooks® are a great bargain. Nine books, packed with everything to teach you digital and analog electronics from scratch.

Bugbooks I & II - 750 pages, with experiments in basic digital electronics to sophisticated circuits. $17.00.

Bugbook IIA - Advanced topics in digital logic transmission. $5.00.

Bugbook III - General overview of microcomputer systems with particular emphasis on interfacing, I/O, programming in machine languages. 597 pages. $15.00.

Bugbooks V & VI - Digital logic and microcomputers from ground zero. Emphasis on self-teaching. No previous knowledge of electronics needed. Over 900 pages. $19.00.

BRS-1 - Lab workbook providing theory and applications of the 555 timer chip. $6.95.

BRS-2 - Text/workbook covering the best design of low-pass, high-pass, bandpass and notch filters. $8.50

BRS-4 - OP-AMP and linear course. 300 pages. $9.00.

Select your $35 (or more) book value and get $6.50 worth of Bugbacks now. Send check or money order directly to E&L Instruments. We'll ship postpaid anywhere in continental U.S.

E&L INSTRUMENTS, INC.
61 First Street, Derby, Conn. 06418
(203) 735-8774 Telex No. 96 3536

Please send the following Bugbooks, postpaid. I've enclosed
$ ______ . ☐ Bugbook I/II $17.00 ☐ IIA $5.00 ☐ III $15.00
☐ V/VI $19.00 ☐ BRS-1 $6.95 ☐ BRS-2 $8.50 ☐ BRS-4 $9.00

Name ______________________

Address ______________________

City __________ State ________ Zip ____

Enclose check, money order or numbers from BankAmericard/ Visa or Master Charge. We will ship postpaid anywhere in Continental U. S.

TM

Reader Interrupt

And now a word from our sponsors. Advertisers like to see where a magazine is headed. So when you write for information or when you purchase a product written up or advertised in ROM, *please mention where you heard about it. In fact, even if you don't see it advertised in* ROM *yet, mention that you read us. The more response you give, the more interest you show, the more the personal computing field—and that includes news about it, ads about it, and articles about it—can be directed to you, our readers.*

After this announcement we return you to our regular program...

Dear ROM,

I had to write to tell you how much I enjoy the crossword puzzles. Being customized for readers just like me, they're especially fun to do. And in tracking down some clues I found I learned a few things that I didn't know! This is the kind of thing that sets you apart from your competition. Keep up the innovating.

B. M. Hanson
Westport, Connecticut

Dear ROM,

In my article "Make Me More Music, Maestro Micro" (November *ROM*), the "E saxophone" you talk about on page 62 should have been "E^b saxophone." Someone using this example and trying to figure out how to transpose will get very confused.

Also in the sentence:

> Transpose the music up (or down) by adding to (or subtracting from) "P" in line 435 for each semitone higher (or lower) you wish to hear it

you left out "higher (or lower)" so the sentence makes no sense at all.

Finally, in the last paragraph on page 62, after two sentences describing how one can alter the PLAY program, you add a few dots and then tell people how they can tune the computer by adjusting Line 560. Unfortunately, Line 560 is in the SCORE program, not the PLAY program! You "edited" this line, taking it from where I had it (after the sentence about transposing) in a discussion of SCORE, and put it for some reason at the end of this discussion of PLAY. To anybody trying to follow me, these changes were illogical, and obscured my meaning.

Dorothy Siegel
Brooklyn, New York

Well, they all laughed when I sat down at the piano—and I guess they still are.

ROMulus

Dear ROM,

I've been wondering what happened to Ted Nelson after *Computer Lib*. I was assuming that such an effort wiped him out as it would have me. However, I'm happy the man has such vitality.

I have been trying to start some kind of computer club here but so far not much interest, generally this is the place to escape such things. However I've got an office downtown. And a telephone. And am expecting to receive a Radio Shack toy computer any day. Actually yesterday. Also I'm ignorant. So here's hoping your magazine's not a myth.

Jon Sanford
Santa Fe, New Mexico

Dear ROM,

I just came across your October issue—borrowed—and decided I have to subscribe. Other computer magazines I buy off the racks at computer stores if and when I find interesting articles, but in the October *ROM* just about every article was interesting.

I think there are a lot of people with, or about to get, computers just like me—I know five—not too knowledgeable or interested in hardware and its design, not very skilled at software, but eager to learn. So *ROM,* with its *ROM*tutorials and selections from a variety of possible topics suitable for someone just past beginning with microcomputers, is the magazine I'll subscribe to, rather than *dr. dobb's, BYTE,* or *Kilobaud,* all of which I read, off the rack, though.

Jerry Mueller
San Francisco, California

ROM
COMPUTER APPLICATIONS FOR LIVING

Editor and Publisher
Erik Sandberg-Diment

West Coast Editor
Lee Felsenstein

Associate Editors
Sue Neillson
Janet C. Robertson

Assistant Editor
Karen K. Jambeck

Contributing Editors
Frederick W. Chesson
Bill Etra
Louise Etra
Sandra Faye
Ed Hershberger
Avery Johnson
Arthur Karshmer
Richard W. Langer
Theodor Nelson
Robert Osband
Eben Ostby
Frederik Pohl
Andrew Singer
Alvin Toffler

Crossword Puzzle Editor
Daniel Alber

Editorial Assistant
Donna Parson

Art Director
Susan Reid

Assistant Art Directors
Cindy Hain
Korkie

Contributing Artists
Steve Gerling
Robert Grossman
Luis Jimenez
Rex Ruden
Linda Smythe

Staff Photographer
Thomas Hall

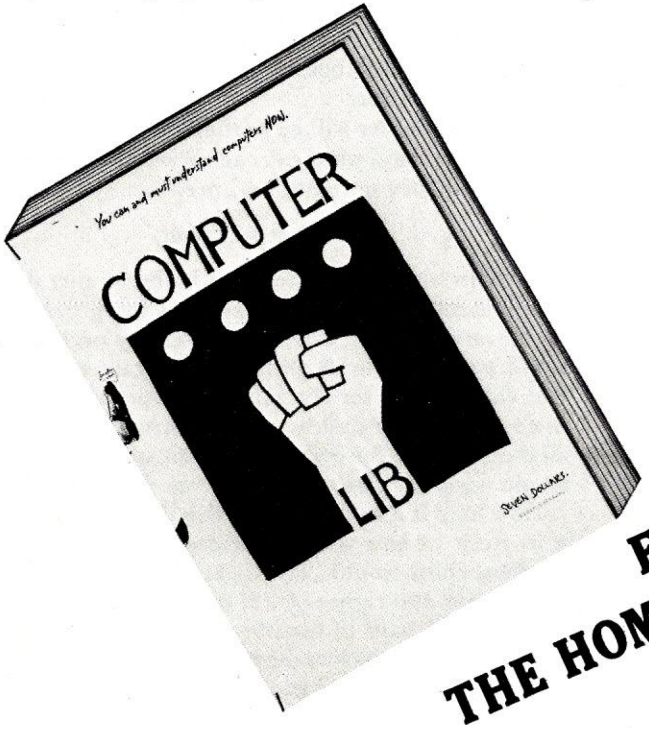
Composition
Lynn A. Archer

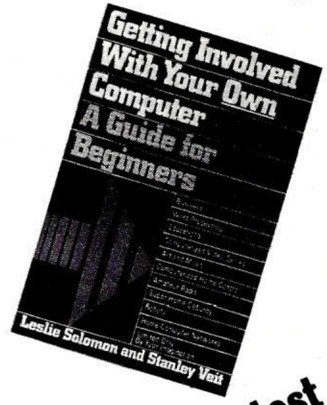
Special Assistant
Jennifer L. Burr

Counsel
Peter Feilbogen

The ROMshelf

For your further reading pleasure . . .




Getting Involved With Your Own Computer
A Guide for Beginners
Leslie Solomon and Stanley Veit

FLASH! The Latest Nelson
THE HOME COMPUTER REVOLUTION
$2.00
Order NOW!

Getting Involved With Your Own Computer—$5.95

Stanley Veit, proprietor of the Computer Mart of New York, the first computer store east of the Mississippi, and Leslie Solomon, Technical Editor of **Popular Electronics** have put together a solid beginner's guide to personal computers. All the basics you'll need to pick a system, and enough background material to get you thoroughly hooked.

Computer Lib/Dream Machines—$7.00

This is the classic "Whole Earth" style catalogue of fact and futuredreams that has probably launched more computerists on the road to hardware insolvency than any other single volume. You'll either love it or hate it—but you'll never find it boring. A great gift for introducing a friend to the world of personal computing.

On press now. For all those who have been waiting for Ted Nelson's new book, here it is. And it's real vintage.

Contents

1. **Home Computers!**
2. **The Magic of the Interactive Computer**
3. **What You Can Do Now**
4. **How We Got Here**
5. **What It Is**
6. **Programming**
7. **How To Get Started**
8. **The Art of the Computer Screen**
9. **A New Kind of Mental Life**
10. **The Future**
11. **Great Issues Involving Computers**
12. **The End of the Myth**
13. **Personal Digital Services**
14. **Those Unforgettable Next Two Years**

Please send coupon to:

The ROMshelf
Route 97
Hampton, CT 06247

☐My check is enclosed.
☐Master Charge #__________ Exp. date________
☐BankAmericard/Visa #__________ Exp. date________

Name____________________
Address____________________
City________ **State**______ **Zip**__________

Paperbacks from the ROMshelf

The Home Computer Revolution by Ted Nelson	$ ______
Getting Involved With Your Own Computer: A Guide for Beginners by Leslie Solomon and Stanley Veit	$ ______
Computer Lib/Dream Machines by Ted Nelson	$ ______

Please add fifty cents per book for postage and handling.
Total $ ______

Please allow 4-6 weeks for delivery. Feel free to photocopy this page if you wish to keep your ROM intact.

Missionary Position

THE ART OF THE COMPUTER SCREEN

by Theodor Nelson

The interactive computer screen will be mankind's new home.

The sooner we understand it, the better.

Your Interactive Graphic Screen

You can get graphical screens for your personal computer already. Most of the prebuilts are being offered with graphic screens; you can put them on the others as accessories.

People want them for games. People want them for practical uses. And people want them for sheer excitement. With this equipment—and suitable programs—you can make your own cartoons, your own interactive pictures, your own complete console for living.

But so far the programming to be seen on hobbyist screens has been rudimentary and difficult. (For instance, every time you see a Video Dazzler, you generally see Steve Dompier's same little picture of a champagne bottle pouring.) There are few interactive animations for these systems, as yet.

Just what are we talking about?

The Commodore PET offers a screen with text and certain picture capabilities. Short line segments, vertical and horizontal, can be combined into pictures or animations. Patterns of dots may also be put on the screen, but in certain very restricted arrangements.

The Radio Shack computer allows a certain pictorial capacity with little squares, 48 (vertical) by 128 (horizontal). Separate TV required.

The Merlin video board for S-100 machines allows graphics of 96 by 128 squares. Separate TV required.

The Video Dazzler from Cromemco offers color graphics of 64 by 64 squares in eight colors. This is also an S-100 system. The Super-Dazzler, still in the works, promises much higher resolution, but we don't know when. Separate TV required.

The Levine Board (available from the Itty Bitty Machine Co., Evanston, Illinois) is an S-100 board offering 256 x 192 squares of graphic animation. Unlike the Dazzler, it does not slow the computer down. Separate TV required.

The Compucolor machine, a prebuilt with color video included, offers graphics in color, 192 x 160 boxes. This has certain peculiarities, restricting the display to only two colors within small regions. But the machine is inexpensive at $3000, considering all it does.

These are only a few of the many fabulous pieces of equipment *now* on the amateur-computer market, offering different kinds of interactive pictorial capability. We won't even get into the programming problem. But we will talk about what it's for.

More and better will be available soon. The thing to do now is understand what you can *do* with the screens, understand what they portend, and prepare.

What's Coming

Perhaps what will matter most in the coming decade will be the design of interactive systems for people to use in their everyday lives. These will resemble nothing so much as video games; but they will be video games about real life and video games for the mind. Tomorrow's desk, tomorrow's automobile dashboard, tomorrow's control panel—all these will use the computer screen as a magic viewer and magic wand; a gateway to what we want to see or do.

How hard it is to write about this in a column! If you saw it in front of you you'd understand it immediately—the smallest child would. Five years from now you'll see it everywhere. But right now, at this instant, the brink of the new world, I have to fumble with words.

Earlier we saw how easily a computer can be made to behave interactively. The general principle is this: something appears on the screen, typed by the computer; you type something back (take your time); the machine replies at once with something new.

The Most Important Computer Program Ever Written

All the computer-screen systems of tomorrow were foreshadowed by one astonishing program created by an isolated genius in the early sixties.

A stern, thoughtful young man named Ivan Sutherland, then a graduate student at MIT, was given permission to use the special graphics computer at Lincoln Laboratory.

Lincoln Laboratory is a stern, thoughtful complex on the outside of Boston where they do electronics research associated with warfare. The special graphics computer was the TX-2, built especially for experimentation with pictures on computer screens. What did this have to do with war research? Only that the military finds out about new developments first, and so that is where computer screens got their first boost.

Ivan Sutherland, in any case, showed a rare vision in what he chose to do with the TX-2 computer—and how he did it.

He created a system that allowed you to draw on the screen. For this reason he called his program SKETCHPAD.

The SKETCHPAD program allowed you to draw on the computer screen as you might on paper—but with remarkable new capabilities.

You could draw a picture on the screen with the lightpen—and then file the picture away in the computer's memory. You could, indeed, save numerous pictures in this way.

You could then combine the pictures, pulling out copies from memory and putting them amongst one another.

For example, you could make a picture of a rabbit and a picture of a rocket, and then put little rabbits all over a large rocket. Or, little rockets all over a large rabbit.

The screen on which the picture appeared did not necessarily show all the details; the important thing was that the

---

From *The Home Computer Revolution,* Copyright © 1977 by Theodor Nelson. *All Rights Reserved.*

Every computer needs a ROM - so do you!

SOL: THE INSIDE STORY by Lee Felsenstein
ALTAIR AND THE ART OF MOTORCYCLE SHOP MAINTENANCE
TOOLS by Joseph Weizenbaum
ROM
COMPUTER APPLICATIONS FOR LIVING
MEMORY, MEMORY, HOW MUCH MEMORY? by Stan Veit
APLomania by Eben F. Ostby
BINARY CLOCKS IN 0011/0100 TIME
ROM
COMPUTER APPLICATIONS FOR LIVING
BIOFEEDFORWARD by Bill Etra
COMPUTER WRESTLING: THE PROGRAM OF CHAMPIONS
XEROXES & OTHER HARD COPY OFF YOUR CRT
A PAYROLL PROGRAM FOR YOUR SMALL BUSINESS
ROM
COMPUTER APPLICATIONS FOR LIVING
THE WONDERFUL DREAMS OF DR. K
SCOTT JOPLIN ON YOUR SCI-FI HI-FI
ROM
KILOBYTE CARD Memory for Pennies
8080
PROJECT PROMETHEUS by Lee Felsenstein
HOW DOES YOUR CRYSTAL GROW by Sandra Faye
MAKE ME MORE MUSIC, MAESTRO MICRO by Dorothy Siegel
ROM
COMPUTER APPLICATIONS FOR LIVING
THE WORDSLINGER 2200 Characters Per Second

The computer magazine for the curious

Computers Challenge America's Cup by Eben Ostby ☐ *A Beginner's Guide to Peripherals: Input/Output Devices Your Mother Never Told You About* by Leslie Solomon and Stanley Veit ☐ *Computer Country: An Electronic Jungle Gym for Kids* by Lee Felsenstein ☐ *The Best Slot Machine Game Ever* by Tom Digate ☐ *The Micro Diet: Better Health through Electronics* by Karen E. Brothers and Louise L. Silver ☐ *Come Closer and We Won't Even Have to Talk* by Avery Johnson ☐ *The Kit and I, Part Four: Testing, Testing* by Richard W. Langer ☐ *Computer Models in Psychology* by Joseph Weizenbaum ☐ *Micro, Micro on the Wall, How Will I Look When I Am Tall?* by Stuart Dambrot ☐ *Copycat Computer* by Tom Digate ☐ *Talk Is Cheap* by Hesh Wiener ☐ *Project Prometheus: Going Solar with Your Micro* by Lee Felsenstein ☐ *BASIC from the Word GOTO* by Eben Ostby ☐ *Chipmaker, Chipmaker, How Does Your Crystal Grow?* by Sandra Faye ☐ *The Kit and I, Part Three: Personality Plus* by Richard W. Langer ☐ *Make Me More Music, Maestro Micro* by Dorothy Siegel ☐ *Wings in Wind Tunnels: Computer Models and Theories* by Joseph Weizenbaum ☐ *What Is a Microcomputer System?* by Leslie Solomon and Stanley Veit ☐ *Maintaining Your Micro* by O.S. (The Old Soldier) ☐ *Time Sharing on the Family Micro* by Barry Yarkon ☐ *The Wordslinger: 2200 Characters per Second* by Stuart Dambrot ☐ *Light Fantastic: The Kinetic Sculpture of Michael Mayock* by Tom Moldvay and Lawrence Schick ☐ *From Bombs to ROMs* by Lavinia Dimond ☐ *Guard against Crib Death with Your Micro* by Jon Glick ☐ *Home Computers: The Products America May Never Know It Needs* by Martin Himmelfarb ☐ *Putting Two and Two Together* by Tom Pittman ☐ *The Wonderful Dreams of Dr. K* by Hesh Wiener ☐ *The Kilobyte Card: Memories for Pennies* by Thorn Veblen ☐ *The Unlikely Birth of a Computer Artist* by Richard Helmick ☐ *Scott Joplin on Your Sci-Fi Hi-Fi* by Dorothy Siegel ☐ *Building a Basic Music Board* by Eben F. Ostby ☐ *The Compulsive Programmer* by Joseph Weizenbaum ☐ *The Very Best Defense (a short story)* by Laurence M. Janifer ☐ *Chart Up and Flow Right* by Eben F. Ostby ☐ *Computer Wrestling: The Program of Champions* by Lee Felsenstein ☐ *Forget Me, Forget Me Not* by Avery Johnson ☐ *PLATO Makes Learning Mickey Mouse* by Elisabeth R. Lyman ☐ *Charged Couples* by Sandra Faye Carroll ☐ *Xeroxes and Other Hard Copy off Your CRT* by Bill Etra ☐ *The Kit and I, Part Two: or Power to the Computer* by Richard W. Langer ☐ *How Computers Work* by Joseph Weizenbaum ☐ *Personally Yours from IBM* by Eben F. Ostby ☐ *A Payroll Program for Your Small Business* by Robert G. Forbes ☐ *Memories Are Made of This* by Lee Felsenstein ☐ *Memory, Memory, How Much Memory?* by Stan Veit ☐ *Software—The Genie in the Bottle* by Tom Pittman ☐ *Your Computer or Your Wife* by Susan Gilpatrick ☐

Every monthly issue keeps you abreast of the latest microcomputer applications for home, school, and office. Written by professionals who know how to present microcomputing in a lively, readable, and understandable fashion, ROM is fun. ROM is instructive. ROM is everything you ever wanted in a computer magazine.

Look what you've been missing without your monthly ROM!

Plus columns by Ted Nelson, Andrew Singer, Bill Etra, and A.I. Karshmer, on Artificial Intelligence, The Future, The Human Factor in Computing.... Plus practical software, listings, documentation, new peripherals, interfaces, games.... And more, much more.

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97, Box R
Hampton, CT 06247

Name

Address

City **State** **Zip**

U.S.A.: ☐One year $15 ☐Two years $28 ☐Three years $39
Canada & Mexico: Please add $2/yr. additional postage
Europe & South America: Please add $12/yr. additional postage
All other continents: Please add $24/yr. additional postage
☐Check/money order encl. ☐Master Charge ☐BankAmericard
Exp. date **Card#**

Please allow 4-6 weeks for delivery.

details were *in* the computer; when you magnified a picture sufficiently, they would come into view.

You could magnify and shrink the picture to a spectacular degree. You could fill a rocket picture with rabbit pictures, then shrink that until all that was visible was a tiny rodent; then you could make copies of *that,* and dot them all over a large copy of the rabbit picture. So when you expanded the big rabbit till only a small part showed (so it would be the size of a house, if the screen were large enough), then the foot-long rockets on the screen would each have rabbits the size of a dime.

Finally, if you changed the master picture—say, by putting a third ear on the big rabbit—all the copies would change correspondingly.

The drawing operation in SKETCHPAD was very special. The user would point with the lightpen at a starting-point on the screen, and draw a line from that starting-point to any other position. A line would extend from that position to the tip of the lightpen, and when the lightpen moved, so would the line, stretching like a rubberband from its starting-point. This was called a "rubberband line;" it allowed the user to try out different positions without erasing.

Then, when the user wanted to join two lines, there was a way of *attaching* them: two lines that were attached remained attached, even when the user decided to move one of them.

One of the most important aspects of SKETCHPAD was this: working on a screen, you could try out things you couldn't try out as a draftsman on paper. You were concerning yourself with an abstract version of the drafting problem: you didn't have to sharpen any pencils, or prepare a sheet to draw on, or use a T-square or an eraser. All these functions were built into the program in ways that you could use through the flick of a switch or the pointing of the lightpen. And the drawing itself existed in an abstracted version, that could be freely changed around with no loss of detail.

Thus SKETCHPAD let you try things out before deciding. Instead of making you position a line in one specific way, it was set up to allow you to try a number of different positions and arrangements, with the ease of moving cut-outs around on a table.

It allowed room for human vagueness and judgment. Instead of forcing the user to divide things into sharp categories, or requiring the data to be precise from the beginning—all those stiff restrictions people say "the computer requires"—it let you slide things around to your heart's content. You could rearrange till you got what you wanted, no matter for what reason you wanted it.

There had been lightpens and graphical computer screens before, used in the military. But SKETCHPAD was historic in its simplicity—a simplicity, it must be added, that had been deliberately crafted by a cunning intellect—and its lack of involvement with any particular field. Indeed, it lacked any complications normally tangled with what people actually do. It was, in short, an innocent program, showing how easy human work could be if a computer were set up to be really helpful.

As described here, this may not seem very useful, and that has been part of the problem. SKETCHPAD was a very imaginative, novel program, in which Sutherland invented a lot of new techniques; and it takes imaginative people to see its meaning.

Admittedly the rabbits and rockets are a frivolous example, suited only to a science-fiction convention at Easter. But many other applications are obvious: this would do so much for blueprints, or electronic diagrams, or all the other areas where large and precise drafting is needed. Not that drawings of rabbits, or even drawings of transistors, mean the millennium; but that a new way of working and seeing was possible.

The techniques of the computer screen are general and applicable to *every*thing—but only if you can adapt your mind to thinking in terms of computer screens.

It should be obvious that you can use the techniques of computer screens to do bookkeeping, writing, design, architecture; to plan how to move your furniture, to catalog your goldfish. Whatever your field, whatever the kind of data, you can use the computer screen to store, retrieve, choose, draw, rearrange, correct, adjust; to see instantly the results of an idea, and change the idea accordingly; to enact your work, and see it whole, rather than guess at its consequences and work with little pieces.

This is, of course, completely the opposite of "the computer" that so many people think of: cold-blooded, demanding, and requiring everything people tell it to be set up in difficult codes.

The Failure To See

In the fifteen years since SKETCHPAD, no initiatives worth discussing have been taken by the computer industry to bring us closer to a world of computer screens for everyone. It was not in IBM's economic interest to make computers easy to use, but to sell complication and make it sound necessary. The computer companies, mostly following like goslings after IBM, have simply brought out smaller computers and cheaper terminals. (Screens have finally appeared, but merely because it has become cheaper to put out a terminal with a screen than a terminal that prints; but most screens show no pictures.)

The brainlessness of the ordinary computer companies has now become plain, however; for personal computing has arrived with a bang, and with it the certainty, for all to recognize, of a computer-screen future.

Most people have not seen SKETCHPAD, or the movies of it, and nobody was motivated to tell them. Even many people in the computer field, technically-minded and preoccupied with their own areas, have failed to see the revolutionary implications of these developments. Indeed, many see computer graphics as worthless frivolity, rather than what it is: the beginning of a new world.

In the meantime, the hundreds of young people who have seen what would soon be possible with computer screens have retreated to the universities, or elsewhere, to wait out the situation.

And of course the public has hardly heard of it at all.

Of course most people are not yet prepared to think in terms of computer screens. There is some wrench, some about-face required, much like that of learning to live with the printing press, or the telephone. But for many it will only take five minutes of real interaction to see what's coming, and start thinking about what *they* want.

Some Important Screen Systems

The computer screen is something new on earth. That few people have seen how to use them, or seen how immense will be their impact on society, should perhaps be forgiven. People didn't know what they had on their hands when movies were first invented, either. (I've discussed this in "Getting It Out of Our System," in *Critique of Information Retrieval,* edited by Schechter and published by Thompson Books, 1968.)

But a few dazzling examples have begun to show us how computer screens should be used.

SKETCHPAD showed us what could be done at the screen with pictures. Another system, NLS, has shown what can be done with text.

Douglas Engelbart's "NLS" system, created at Stanford Research Institute, allows a user to read from screens and write on screens, instantly pulling to the screen whatever he wants from large quantities of stored text—or putting new things away.

The many users of Engelbart's system can share the writings that are stored in it, and even make marginal comments on each other's work—all stored electronically.

The only drawback of NLS—aside from its presently high cost—is that it is not for beginning users. To learn its use takes ten days, not ten minutes. The kind of performance it offers is terrific; later systems of this kind will have to be simpler for most people to use. But Englebart has shown the way.

The third spectacular example is Alan Kay's "Dynabook" at Xerox Palo Alto Research Center. The Dynabook is simply a small computer with screen, keyboard, and SMALLTALK language. But the dazzling screen manipulations—pictures, animations, fancy text—are exciting to everyone.

A fourth example is PLATO. The PLATO system, created by Donald Bitzer at the University of Illinois (and now being sold by Control Data Corporation), allows a thousand users, all over the country, to have highly interactive computing and graphics on super-looking graphic screens. (See "PLATO Makes Learning Mickey Mouse" by Elisabeth R. Lyman in *ROM,* September, 1977.)

PLATO costs far too much, and is in its present form a dead end, since it uses an expensive central computer instead of little private computers, like Dynabook; but it remains the most publicly visible system for the human use of interactive computers.

The Anatomy of the Computer Screen

The computer screen is something new on earth, and so we are just discovering—and inventing—its nature.

What to use it *for* is obvious: everything. But how to design overall systems is another question. It can be very hard to do well.

However, the different things people have been putting on the screen can be described and categorized, together with their uses so far.

A *cursor* is a movable marker on the screen. When you control a cursor, it serves to tell the computer program what you are pointing at. When the program controls a cursor, it is a way of showing you what you should be looking at, or where the next thing you type will appear. (The Latin root of "cursor" means runner, and the cursor does indeed run around the screen for you.)

A *menu* is a list on the screen of things the computer is ready to do for you; and if you point at one of the items on the menu, the computer then does it. If there is a dot of light to point at, that is called a *lightbutton.* If the menu is composed of symbols or pictures to point at, it is a *symbol menu.*

A *menuplex* is the complex of menus a user may weave through.

Often a screen will be divided into sections having different functions or activities going on. These are called *panels* or *windows.* A place set aside with no borders is simply an *area.*

If advice appears as to what you may do next, it is called a *prompt.* If an area is set aside for prompting, it is the *prompt area.*

Some systems expect you to type whole commands in, and leave an empty line for the purpose at the top or bottom. This is the *command line.*

Sometimes a symbol on the screen will indicate what is going on; when something else begins, it changes to another symbol. This is a *ding-dong.* (If a cursor changes shape depending on what's happening, this is a *ding-dong cursor.*)

Pop-ins are symbols that appear out of nowhere under certain conditions.

A *peekaboo* is something that appears on the screen if you touch a smaller symbol (the *doorbell*).

These names, of course, give no flavor as to what you can do with them.

Just for an example, let's invent a console for a musician: someone who gives live performances, and plays a piano-type keyboard. Let's call him Irving. We'll call the system SAM, or System for Augmented Music.

Very well: a piano-like keyboard, for input.

The keyboard connects to a central small computer, which actually generates the sounds. Probably there are several computer chips; one to handle all the timing and switching and screen-work, several more to create the tones. (Making tones by computer chip is now becoming cheaper and simpler than having a whole music synthesizer, which has to be wired up specially.)

There are loudspeakers: let's be generous and say eight.

And there is the screen, just above the keyboard. A lightpen dangles before it, ready to be pointed more specifically. Irving will press a footswitch when he wants to tell the computer to act on what he is pointing at.

Irving the musician sits down at his instrument. On the screen, in the main panel, is a menu of voices he may want to play in, like organ-stops. Besides the usual names, like FLUTE and DIAPASON, he also has voices called BAUTANT, TWEEDLE, GRUNDOON, and SNAZ—voices he created through the screen.

With lightpen Irving now selects the name of the voice he wants to play in, BAUTANT. That name now appears on a top reminder line, saying that this is the voice he is playing in.

But more: at the bottom of the screen appear some pop-ins, a miniature map of the loudspeakers. Aiming his lightpen between the speakers on the map, he tells the machine where he wants the sound to appear to be coming from: in this case, the center of the room.

And he plays for awhile.

Now he decides to change the sound. Pausing for a moment, he touches a doorbell next to the word BAUTANT in the main panel. A diagram of the sound appears; swiftly he modifies that diagram. He lets it go, releasing his foot on the pedal; the diagram disappears, but he is playing now in the newly modified sound.

(Note that this part of the facility actually exists in Alan Kay's office at Xerox.)

Now suppose Irving wants to play an orchestral piece with himself (like Mike Oldfield's "Tubular Bells").

Basically it works like this.

As Irving plays on the keyboard, SAM "notes" the timing and pressure of each key-pressing. The timing is noted to the thousandth of a second, the touch about as subtly. Thus an accurate recording is made of what keys were pressed when; this is recorded by the computer as a list of symbols.

This list can be used to replay music just as if it were coming in live. Irving merely touches a lightbutton labeled, "Play It Again, SAM."

And as the computer replays each voice, Irving adds yet another "instrument" to the swelling orchestra—chosen from the voices listed on the screen.

Naturally, each of these instrumental contributions can be modified later if he doesn't like it.

Note that this is not exactly a canned recording. Each of the separate instrumental contributions can be left out, and Irving can replace it with a live performance.

This is something like having many synchronized tape recordings: except that each one can be modified, changed in its sound, or changed in its apparent location—all through the screen.

This is just an example. We could design panels, menus, symbols in great detail, but there's no point right now. These machine functions were just chosen off the cuff; any other things you might want a machine to do can be handled as easily. (But note that a number of computer musicians are building systems for themselves that are rather like this one—including Carl Helmers, the editor of *BYTE* magazine.)

Today, screen-facilities like these are so expensive and esoteric as to be available only to our air traffic controllers, utility companies, and war-control centers. But as the costs go down (and the programming becomes easier), we will have graphical computer consoles for everything.

Consoles for writing, for making music, for communications switchboards, for executives making telephone calls; consoles for artists (that's right), moviemakers, newsmen; for darkroom work, pottery, origami, woodcarving.

Basically they will all have computer, keyboard, screen, disk memory. The interconnections to the outside world will vary, and hence the cost.

But they will use menus and panels and the other things we have mentioned. No systematic study has ever been made of the art of such layout, the menus and symbols and their relation to what you want to do. The closest book so far is James Martin's *The Design of Man-Machine Dialogues,* which treats this study as a form of engineering, not an art.

Views

If something is in a computer system, there must be a good way to view it on a computer screen. There may, indeed, be some new and special way.

Since programs can be created to zip through stored data and analyze it in various ways, someone who is concerned with a particular form of data naturally has an interest in creating viewing-programs specially suited to those concerns.

For instance, text.

Someone interested in text naturally wants to run it forward and back on the screen, meaning up and down, at great speed; to be able to see all the headings, and from the list of headings to jump to the text beneath any one of them, just by pointing.

(Sophisticated users will probably need text systems with a much more elaborate structure, however; see *Computer Lib.*)

If you are interested in such things as census data—complicated boxes of numbers—the computer can be programmed to analyze it into all kinds of statistical breakdowns: numerical tables highlighting various aspects.

But wait! Why be satisfied with numerical tables? The graphical screen can be easily programmed to give you bar charts, pie diagrams, diagrams in proportional shades of grey. Or even new kinds of diagrams that can be rotated in multiple dimensions, presenting to the eye things you could never see before.

Then consider maps.

When the computer stores maps, it can store them in new forms. Through the screen you can magnify the map from the entire nation down to an individual street, if the information is there; no, down to the fine print on a chewing-gum wrapper in the gutter, if *that* information is there.

Map data is two-dimensional. But the computer can also hold information allowing it to present three-dimensional scenes.

Some screen-systems show a three-dimensional object as a system of lines—as in *Star Wars,* where the map of the Death Star, in three dimensions, is brought to the good guys just in time by Artoo Detoo. The three-dimensional line-drawn map in the briefing was in fact created on just such a system, on *our* planet.

Such three-dimensional mapping will become of increasing importance, especially in architecture, research, and teaching.

But once you have three-dimensional data—that is, information precisely describing the coordinates of spatial objects—it need not be viewed as lines only. Certain very expensive viewing-systems permit you to see it as a *colored photograph,* showing exactly how such scenes or objects would appear to a living viewer. And this offers the advantage that you need not build the object physically to visualize it, or view it, or photograph it. You need only create the data structure that represents it in the computer system.

NASA has used this approach very successfully, to make "photographs" of what certain complex space equipment would look like if they built it. This way both Congressmen and engineers can be sure they're talking about the same thing.

Soon, it will be possible to do trick visual effects like the big ones of *Star Wars*—great rockets, planets, monsters, scenery, what have you—without having either models or made-up actors. It will only be necessary to create a computer representation of the desired stuff, and the computer will make the movie or the visual insert, frame by frame.

Finally, one clever engineer thinks he can put this all in your home or school. The big fancy systems for fake photography, the kind you'd use for *Star Wars,* cost a great deal of money, like a million dollars. But Ron Swallow of HUMRRO, a research organization in Alexandria, Virginia, believes he can put it all in a box with a color TV. So instead of your home computer screen merely showing *regular* interactive graphics (and two-dimensional pictures), you can travel through whole worlds—cities and canyons and planets and playgrounds—that look almost *real.* He says the terminal will cost $5000 in a couple of years.

All these different kinds of views will become important. And all will increasingly appear, and become familiar, in different panels of our control screens.

The Frontier: Clarity

Many people seem to think that bigger and better complications mean progress in computers.

They are totally wrong.

Beyond the Computer Screen

Anything you want to do with information can be done at a screen; soon it can probably be done better there.

For instance, if your screen is connected to a good text system and sufficient memory, you can certainly do better writing there than is possible with a typewriter. (Unfortunately, there are as yet few good text systems—but there will be more soon.)

Outside Control Diagrams

Yes, for handling information the computer screen is tops. But it has a more portentous capability still.

You will recall that computers can be hooked up to any other machine that can be controlled electronically. Thus a computer program can control a gas pump, a rotisserie, an oil well.

But in turn, *you,* at a computer screen, can direct the computer to take action in the outside world, making it turn on an eggbeater, or a drawbridge, or a stereo. By adjusting a picture to what you want.

A diagram that controls events—in the computer itself, or in the outside world—is a *control diagram.* If the diagram controls things outside the computer, it is an *outside control diagram.*

Control diagrams can be used, as we have seen, to control the operation of your computer itself. Whatever you want to do with a computer can ultimately be done most easily with control diagrams. But control diagrams are a powerful way to work with the outside world as well.

A practical application of outside control diagrams: there are now oil refineries where nobody goes around turning valves by hand any more, when the petroleum is supposed to take a new route.

Instead, an operator studies a map of the refinery on the screen. Selecting an area of the refinery where he wants to reset a valve, he touches that part of the screen with his lightpen; that area expands to fill the screen. He keeps expanding the map, and more details come into place, until he sees the valve he wants—the magnification is now sufficient to show it. With the lightpen he touches the valve's symbol, and a changing number shows the changing percent of flow.

Satisfied with that one, he changes a dozen more; all in less than a minute.

It's all going to be that way.

There will be setups run by control diagrams for editing movies, for running factories, for opening and shutting down public buildings, for lighting cities.

(You could probably drive your car with a lightpen on a control diagram—but your state Department of Transportation might not think it was safe.)

You should note one difficulty with controlling objects in the world by computer: *it's expensive.* The centralized hookup between the outside and the computer is the hard part, especially if it has to be reliable. The computer itself, and even the program for it, is negligible in cost by comparison.

Clarity and the Design of Objects

Let us briefly digress from the subject of computers, and talk in general about machines that are sold for human use.

Industry persists in turning out badly-thought-out objects that nobody can understand.

The technical things that consumers buy, like tape recorders, have always been badly designed. Designers have come out with a chaotic variety of confusing objects, differing widely. Most tape recorders are difficult to use, some ridiculously difficult. Yet tape recorders only do a few simple things; it's their bad design that makes them complicated.

Recent laws have made it mandatory for all contracts involving consumers to be written in simple English. What we need is a corresponding rule for the design of objects and systems for consumers. Just as the criterion for consumer contracts is that they must be readable by the average high school graduate, a corresponding rule for things sold to consumers ought to be that they have to be understandable in less than ten minutes of instruction. This ten-minute rule should be tatooed on everyone who designs consumer products.

Many engineers and technicians have claimed that this can't be done. Balderdash! It is merely difficult. Moreover, it takes intense dedication to clarity, and repeated revision and rethinking. You have to try over and over until a thing gets simple enough, just as you have to try over and over to make writing clear, and just as you have to rearrange over and over to edit a movie just right.

Another reason that technicians do not like the ten-minute rule is that it deemphasizes what they like to do, and minimizes their achievements in their favorite area of operations. Technical people like to think about technical things; that is why they are technical people. (One engineer has confided in me that he is never really happy unless he is *feeling those chips with his fingers.* This is a very poignant admission.) They think that designing a tape recorder, or a computer program for people to use, is a technical matter. It isn't.

Designing an object to be simple and clear takes at least twice as long as the usual way. It requires concentration at the outset on *how a clear and simple system would work,* followed by the steps required to make it come out that way—steps which are often much harder and more complex than the ordinary ones. It also requires relentless pursuit of that simplicity even when obstacles appear which would seem to stand in the way of that simplicity.

Much has to be reconsidered, of course, when it turns out that the simple-and-clear design is not feasible in its premeditated form; after many changes and reconsiderations, it is the brave designer who wins simplicity and clarity out of the tangle of different pressures.

This is not a column about tape recorders; suffice it to say that I have only seen one tape recorder I considered well designed. This was the Sony TC-50. It is no longer available. People think they want a lot of buttons. ▼

AIQuotient

COMPUTER VISION

by
A. I.
Karshmer

What is green on the top, brown on the bottom, and surrounded by green and blue? What is red and yellow on top, brown on the bottom, and surrounded by white and blue? Simple, you say, it must be a tree at different times of the year—and, of course, you're right.

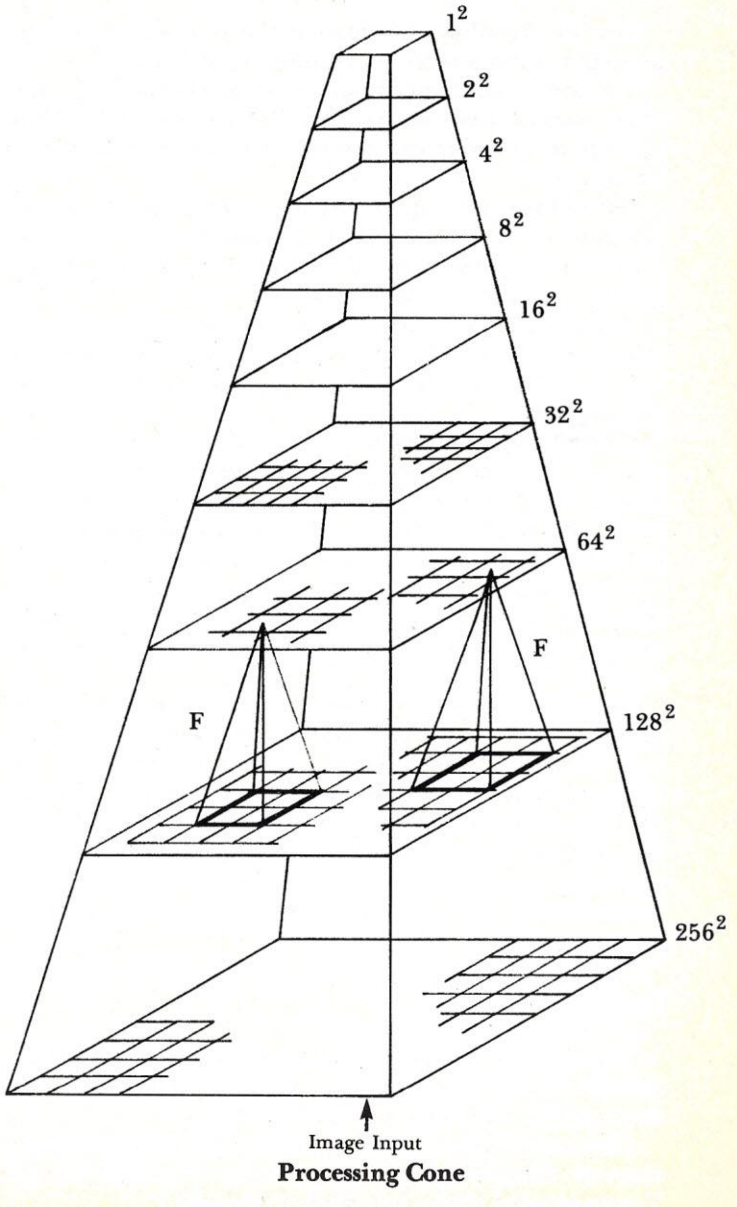
While arriving at this conclusion is almost trivial for you, it is very difficult for a computer to come to the same conclusion. This level of understanding of the natural world is something we take for granted, and without it our visual system would be of little value to us. Vision involves much more than light being focused through the lens of the eye and projected onto the retina at the back of the eye. If simulating human vision on a computer seems complicated to you now, hang on, the problem gets more complex as we ask such questions as "How do we represent a picture in a computer?" and "How do we recognize the objects in that picture?" Questions such as these are typical of the problems faced by that part of the artificial intelligence (AI) community which is interested in creating computers that can "see" and "understand" the natural world.

This month I will present a general overview of a scene-analysis system currently under development by Riseman and Hanson at the University of Massachusetts, and I will discuss how microprocessors might be used in implementation of this system. The University of Massachusetts system implements some of the most advanced techniques currently being used by AI researchers in scene analysis.

The VISIONS system at the University of Massachusetts is composed of two major subsystems, referred to as the low-level system and the high-level system. The goal of the low-level system is the segmentation of a picture into regions representing major portions of the surfaces of objects. In addition, characteristic features are obtained for the regions in the image—such features as hue, saturation, intensity, texture, and shape. The goal of the high-level system is the construction of a three-dimensional model of the objects in the scene represented by the segmented two-dimensional image.

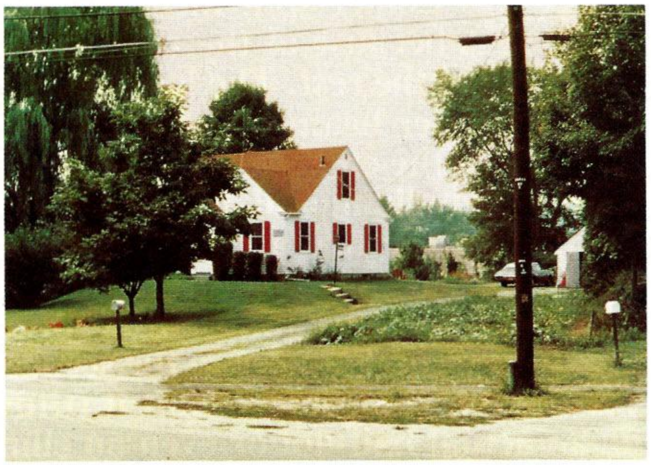
Low-level processing begins with image formation. A color picture is taken of a natural outdoor scene and the picture is digitized with a resolution of 1024 x 1024 picture elements (pixels). Each pixel is represented by three quantities—the red, green, and blue intensities—with each intensity normalized to an eight-bit value. This digitized image is usually averaged to obtain a resolution of 256 x 256, and intensity values are rounded to six-bits. Certain enhancement operators are used to improve contrast and remove "noise" introduced during the digitization phase.

The final task of the low-level system is segmentation. The goal of the segmentation phase is to find the significant regions, lines, and vertices in the image. In the VISIONS system segmentation is accomplished through the use of a mechanism called the processing cone (see diagram), which is a simulation of a parallel array computer which is hierarchically organized. The 256 x 256 digitized image is input to a 256 x 256 array of microcomputers at the lowest level. Each microcomputer can analyze information in a local neighborhood of the pixel on which it is focused. At the next level up in the cone is a smaller array (128 x 128) of microprocessors each of which can analyze information in a 2 x 2 array of processors at the level below, and so on up the cone. Information flows up and down the cone in a parallel fashion. A function may be applied at a given time, t, to a local window at a given level of the cone, and this function may be applied simultaneously in parallel to all local windows across the entire array. The kinds of functions applied are usually edge op-

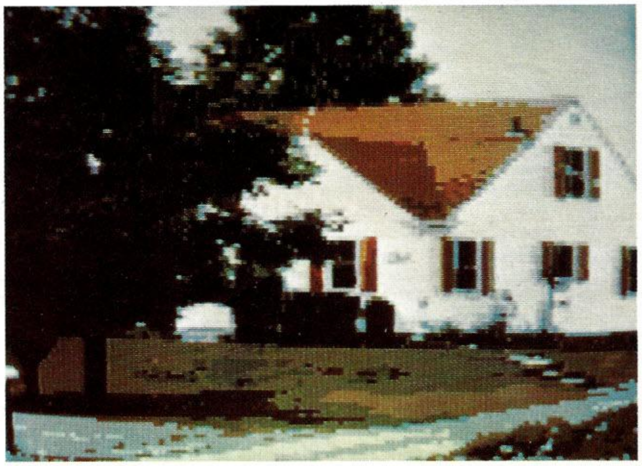


Processing Cone

FOUR STEPS IN VISION LOW-LEVEL SYSTEM



Original Photograph (Input to Digitizer)



Digitized Image (Output of Digitizer, Displayed on a Computerized Color Graphics Display Monitor)

erators, line finders, and region growers. Some of these functions will be discussed when we return to the discussion of the parallel cone later in the article. Currently, the processing cone is being simulated on a sequential computer; however, the plummeting costs of microprocessors may make it economically feasible to actually construct such a parallel array computer in the near future.

As previously mentioned, the goal of the high-level system is to construct an interpretation of the three-dimensional scene. The major components of the high-level system are the long-term memory (LTM), the short-term memory (STM), several specialized processing elements called knowledge sources (KSs), and the model builder (a program embodying the possible interpretation strategies and the control structure).

The long-term memory is a semantic data base which contains the computer's knowledge of the natural world. This knowledge is organized in a hierarchical network structure where the nodes may represent such entities as frames (stereotypical scenes), objects (sky, grass, house, etc.), and the primitive volumes, surfaces, and contours which make up those objects. The edges in the network represent relationships between the entities in the network. For example, garages and driveways generally appear in typical house scenes, and the garage is generally to one side of the house, with the driveway extending out from the garage to a road which runs parallel to the front of the house. All the spatial relationships are encoded in the edges of the network between the nodes for the house, garage, and driveway.

The short-term memory is where the model of the scene is constructed. It is also a layered network structure. At the lowest levels is a symbolic representation of the segmentation of the image produced by the low-level system. The image is represented by a graph in which the nodes represent relationships between these entities, that is, the color, texture, and shape of a region are attached to the node which represents the region.

During the model-building process, knowledge sources analyze the lower levels of the STM network and build network structures (the model) at the higher levels. For example, the shape knowledge source (KS) will examine region shapes and hypothesize the existence of certain surfaces, volumes, and objects in the scene. The perspective knowledge source (KS) will hypothesize distances to objects, object sizes, and orientation of surfaces. The occlusion module helps determine when two regions in the image lie on the same surface in the scene and when they lie on different surfaces and one surface partially occludes the other. Finally, the model builder is the control program which determines the strategy for interpreting the image. For example, it decides which regions in the image to analyze first, how to evaluate diverse hypotheses proposed by the knowledge sources, and when a consistent interpretation has been found.

As previously stated, the processing cone architecture might be realized as arrays of microcomputers each able to compute certain simple functions on local neighborhoods. For instance, at the lowest level each processor in the 256 x 256 array would be able to perform a simple 1 x 2 *differencing operation* on the image. Such a difference operator looks at two horizontally adjacent pixels in the image and computes the difference in intensity between them. Thus each processor must be able to see the intensity at the pixel on which it is focused as well as the intensity at the pixel to the immediate right. In addition, each processor must contain a certain amount of memory in which to store the output of its computation. Processors at the next higher level may then compute the average of the outputs of the four processors below. As a matter of fact, one of the standard edge-finding algorithms performs 1 x 2 differencing at the lowest level and then averages those results up through the cone to the 64 x 64 level, where local line-finding algorithms combine local edges (one pixel in length) to form lines.

Some of the issues which must be addressed when designing such a microprocessor architecture are: (1) the number of processors required, (2) the amount of memory for each processor, (3) the computations to be performed by each processor, and (4) the amount and complexity of the connection logic for determining the local neighborhood which each processor can view. For the processing cone of

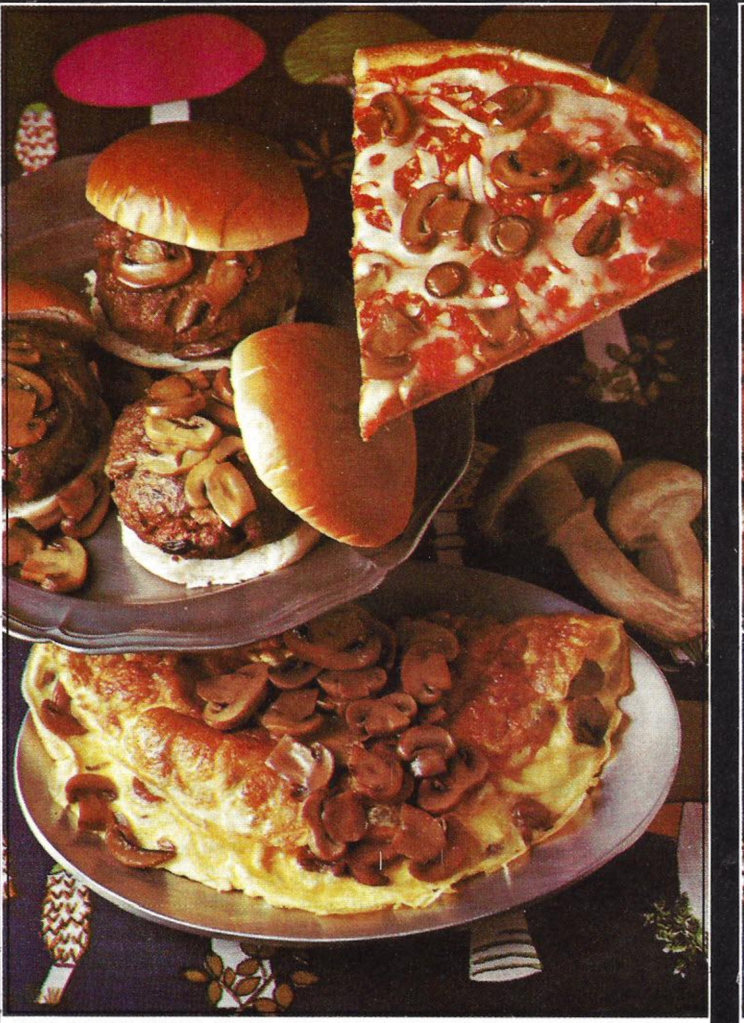
Photographs courtesy of the author

COLLIER IS THE BEST ENGRAVER, PRINTER IN THE COUNTRY.

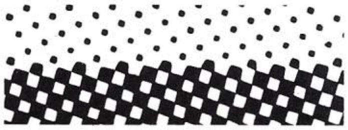
Because. A revolution in engraving has happened. Collier split the dot. And because of the new Laser Beam, Collier is now light years ahead of the rest. The laser does it. Here's how it works. Technically, the laser beam which is used to control film exposure (thus "Program" the engraving) is split into six sectional beams. Each of these is digitally modulated by computer to transfer half-dots of picture information per scanner revolution. Since two picture-information "bits" are available per dot in circumferential direction, it's possible to expose a smaller area in the second "orbit"...or even completely omit it (thus producing an actual half-dot or even an elliptical mini-dot). If that seems to all sound a lot like gobbledygook, don't worry about it. The important thing is, it's here and it does work and it gives your image resolution that you never could get before. We'll be happy to demonstrate it for you. Even happier to produce your next set of four-color letterpress, offset and gravure engravings. Call Collier Graphic Service Company Inc., 240 West 40th Street, New York, New York 10018/(212) 840-0440.

ATLANTA	BOSTON	DETROIT	HARTFORD	NEW YORK	PHILADELPHIA
(404) 892-2383	(617) 965-5660	(313) 259-2111	(203) 367-0706	(212) 840-0440	(215) 988-0110

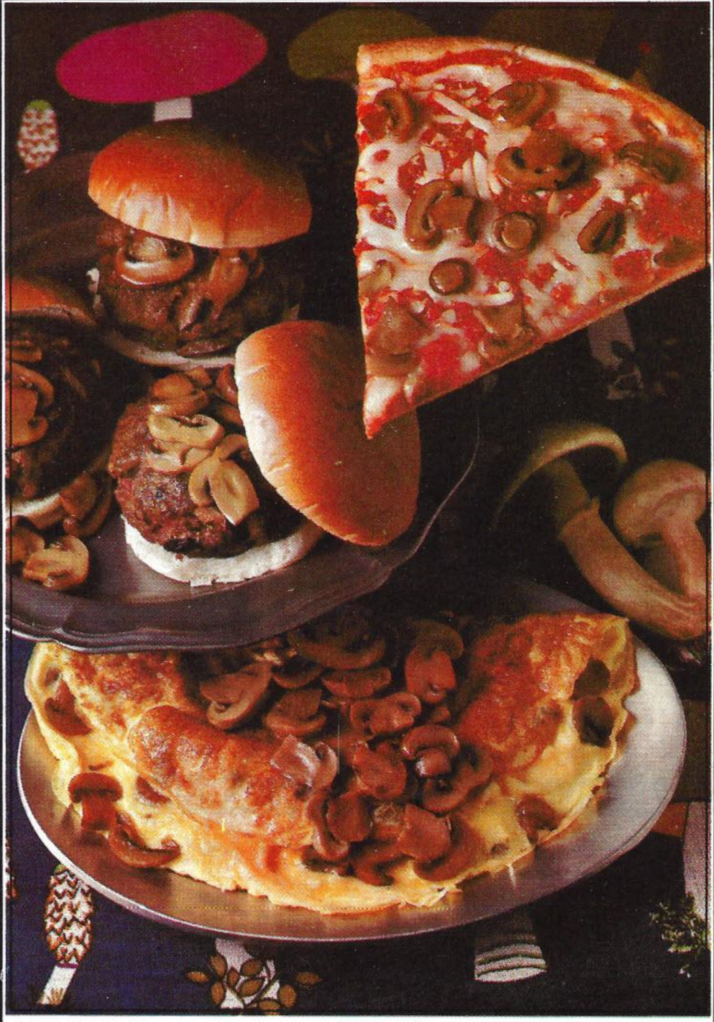
OR CALL TOLL FREE (800) 221-2585/(800) 221-2586



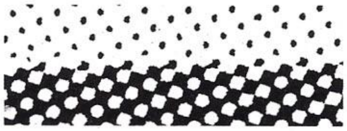
THE ABOVE ENGRAVING WAS MADE WITH THE CONVENTIONAL PLATE MAKING PROCEDURES, AS YOU CAN SEE, IT LACKS COLOR FIDELITY AND SHARPNESS, IF YOU COMPARE IT WITH...



THE CONVENTIONAL ENGRAVING DOT.

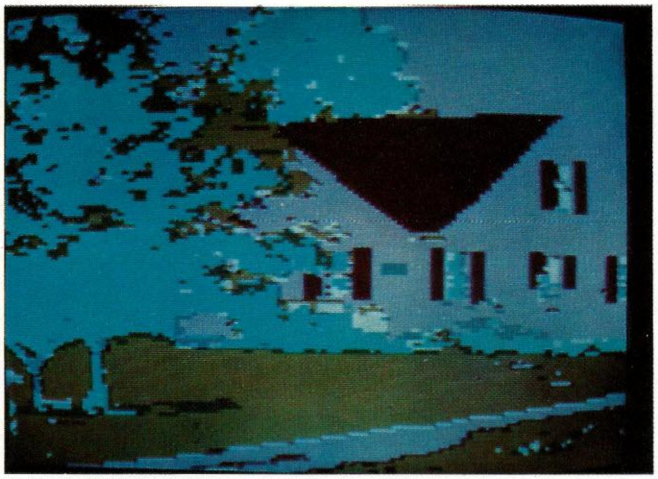


THE COLLIER'S LASER BEAM ENGRAVING, AS YOU CAN SEE FROM THE ABOVE, HAS BRILLIANCE, SHARPNESS AND COLOR FIDELITY THAT ONLY COLLIER'S DOT CAN PROVIDE.

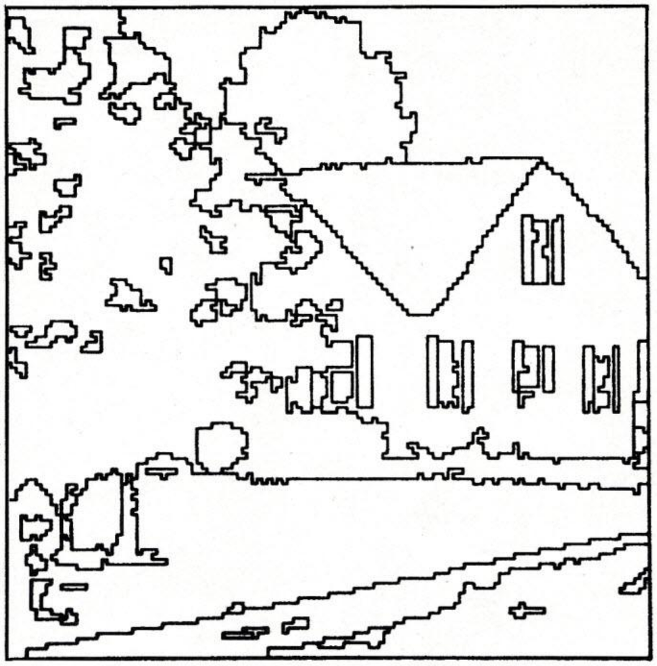


THE COLLIER LASER DOT.

THE COLLIER ADVANTAGE. COLLIER GRAPHIC SERVICES COMPANY, INC., 240 WEST 40TH STREET, NEW YORK, NEW YORK 10018



Computer-Generated Segmentation



Computer-Generated Line Drawing

the VISIONS system, 256 x 256 = 65,536 processors are required for the lowest level; 87,381 are needed in all. Each processor might have access to several operators which perform simple functions such as 1 x 2 differencing, finding the maximum or minimum value in a local window, and computing the average value in a local window. Obviously, the most complex part of the architecture is the interconnecting logic. In the VISIONS system variable size windows are simulated—that is, sometimes each processor looks at the outputs of a 3 x 3 neighborhood of processors below; at other times it may see 5 x 5, 7 x 7, or 9 x 9 neighborhoods. The interconnection circuitry for a fixed neighborhood is obviously less expensive than that for variable neighborhoods; however, fixed neighborhoods limit the computational power of the system. Certainly all of these issues, as well as many more, must be considered before a processing cone may be realized in hardware.

This article has presented an overview of a particular computer vision system for the analysis of natural outdoor scenes. It is considered a general-purpose vision system, and it represents the state of the art for such systems. It should be mentioned that several other computer vision systems are being developed for more specific applications. For example, in the field of medicine there are vision systems which aid radiologists in the analysis of chest X-rays, as well as in the analysis of blood cells. Seismologists are assisted in earthquake fault detection by computer analysis of high-altitude aerial photographs. Major automobile manufacturers are developing vision systems to assist in mechanical assembly. Rapid technological growth and better understanding of visual processes will contribute to a much wider variety of applications in the near future. ▼

ROMtutorial ROMtutorial ROMtutorial

Window: All the picture elements in local neighborhood about a picture element.

Graph: In the context of VISIONS, a graph is an abstract representation of the regions in the image and the relationships between those regions. The regions are represented by "nodes," which may be thought of as "points." The relationships are represented by "edges," which may be thought of as lines between points.

Model: A model is an abstract representation of the scene depicted by the image. It is in the form of a graph.

Knowledge source: A program which accesses some knowledge about natural scenes in the semantic data base and which contains some knowledge about pictures. It uses both kinds of knowledge to make hypotheses about the natural scene based upon information in the picture.

Semantic data base: A data base containing knowledge about natural scenes.

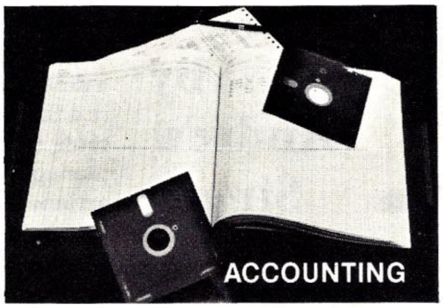
Control structure: A set of rules for selecting alternate paths during program execution.

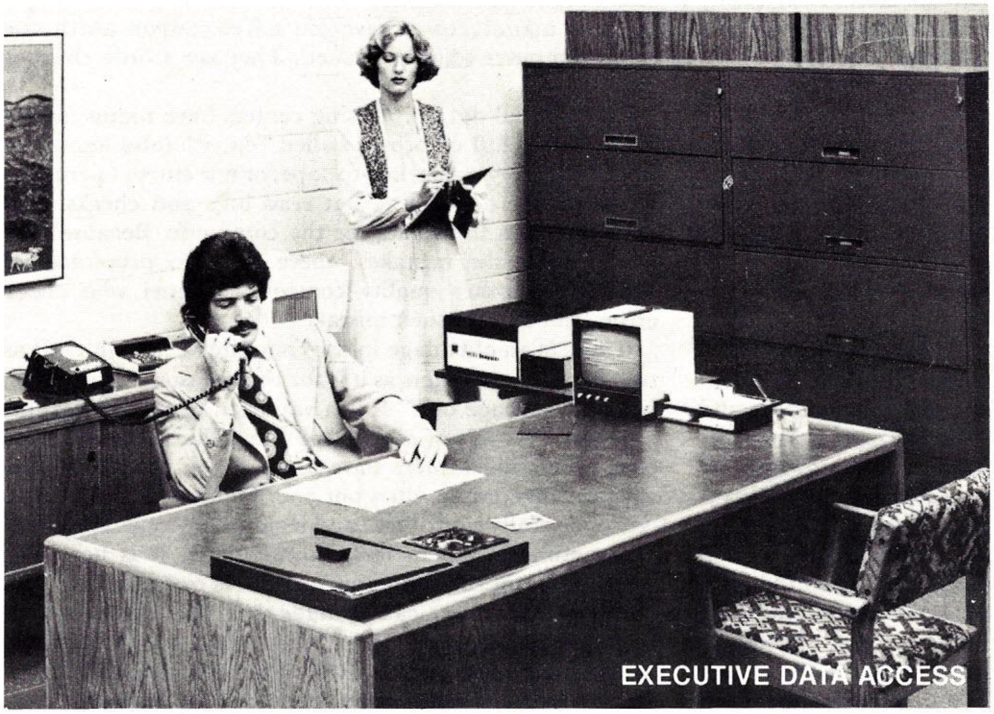
Acknowledgement:

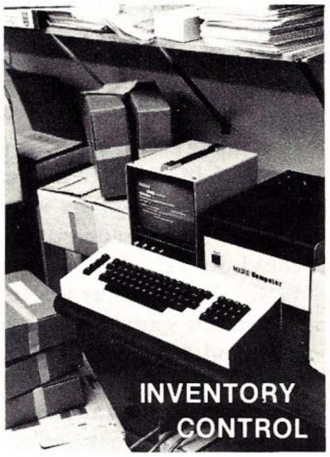
This month's column would not have been possible without the technical assistance provided by Bryant York of the University of Massachusetts VISIONS project.

INTERFACE AGE™ MAGAZINE PRESENTS

MICRO BUSINESS '78™ CONSUMER SHOW







DATE: MARCH 17, 18, 19, 1978

PLACE: PASADENA CONFERENCE CENTER

PASADENA, CALIFORNIA

MICRO BUSINESS '78™ will provide a series of marketing forums and exhibits to introduce the small independent businessman to the new low-cost, high-power business microcomputer that will reduce his company's costs, place him in a more flexible marketplace and provide timely data information.

Emphasis will be on the small budget requirements for purchase of an in-house computer. The show will demonstrate the latest systems, exhibiting complete hardware and software from small hand-held programmable calculators to full turn-key computers.

- Latest in Word Processors
- Newly-Released Business Software
- Low-Cost Text Editing Typewriters
- Modularized Computers

THE LOW COST, dependability, simplicity of operation, and cost savings advantages of microcomputers will be discussed in a series of lectures to remove the many misconceptions the average businessman may have about the microcomputer technology. Lectures by such companies as IBM, Commodore Business Machines and Radio Shack will present the businessman with the latest information about application, service and investment.

Author Adam Osborne will discuss business software.

OTHER LECTURES on the program include:

- Small Business Computing Systems
- Evaluating Your Business Computer Needs
- Software Companies
- The Mainframe Companies & The Small Computer
- The Small Business Computer Company
- Computer Stores and the Small Business System
- Retail Mass Marketing of Microcomputers

Sponsored by: INTERFACE AGE Magazine

EXHIBITORS: PLACE YOUR RESERVATION NOW!

Produced & Managed by:
Show Company International
8687 Melrose Avenue
Los Angeles, California 90069
(213) 659-2050
Ed Tavetian

The Human Factor

WHY NOT DO IT BY HAND?

by Andrew Singer

One of the problems of being a computer consultant is that the people who seek out your advice are invariably determined to use a computer to solve their problems. And often a computer is not the best solution.

Take the case of a friend of mine, a biologist studying lobsters. (A great business if you can get into it because the subjects make such a tasty meal when the experiments are over.) He had accumulated a fair amount of data about lobster behavior that he wanted to "reduce," i.e. make sense of, and he wondered aloud to me whether a computer could do the trick.

"Well," I said, "that would depend on what you've got and what you want to get out of it."

So he and I spent an afternoon looking at it all, and I spent a couple of hours thinking about it all, and I concluded that it would take more time to key in the data and put together the programs to do the computing than the whole job would take by hand. When I told him that, he grimaced and said, "Really?" in a surprised tone. Fortunately, he is a sensible scientist and, after a few glimpses into what was involved, he quickly saw my point.

But I'm afraid he is the exception rather than the rule. Like a sick man who consults another doctor because he doesn't like the diagnosis that the first doctor presented, regardless of how well buttressed by facts, many of my clients look amazed when, as a computer consultant, I tell them, "I think this might be done better manually."

Unfortunately, there are all too many consultants out there ready to tell the computer-hungry that a computer is just what they need. A consultant who recommends a computer (like an attorney who recommends a law suit) is likely to make more money. After all, they are "computer" consultants.

Lately, I've begun to wonder whether I ought to recommend a home computer for some of the applications which people have proposed. Take the home accounting system. Most of the work commercial accounting systems do is called "transaction processing." That is a fancy way of saying that a debit (a bill) or a credit (a check) comes in, is recorded in a file, and is then added to or subtracted from a running total (a balance). As every commercial data-processing manager knows, there's only one way to get all those transactions into computer-readable form—somebody has to key them in. There are machines called "optical scanners" that are capable of reading some printed or typed material directly but these are quite expensive. There used to be a joke around about optical scanners that after you bought one, the manufacturer gave you a free coupon worth one small computer when redeemed. They are a little cheaper now.

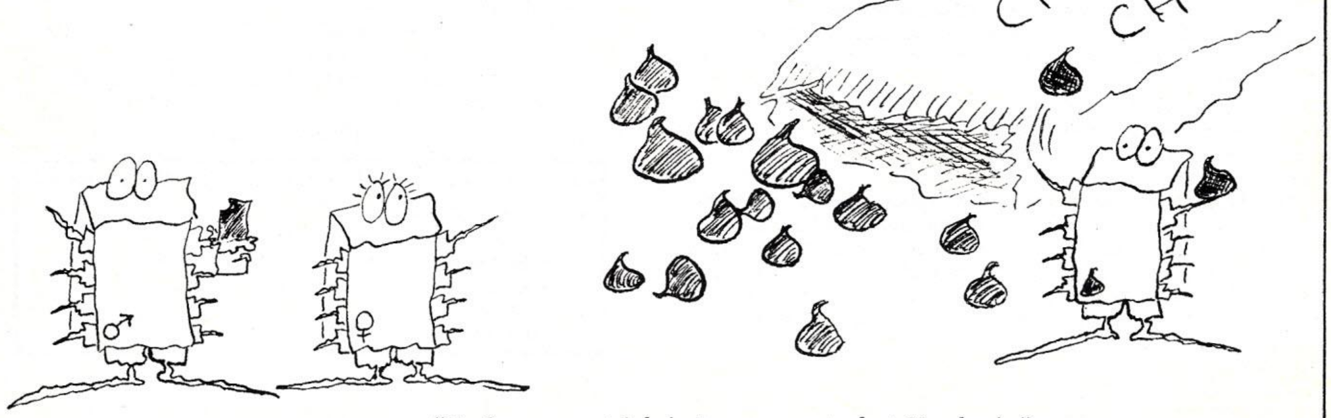
Commercial data-processing centers have rooms, sometimes floors, full of people called "entry" (also known as key-to-disk, keypunch, key-to-tape, online entry) operators. These people do nothing but read bills and checks, and then key them in as data for the computer. Because they sometimes make mistakes, there are also proofreaders, verifiers, or other quality control personnel who check errors and correct such mistakes.

There is an old adage in the computer field, which is as true for tiny computers as it is for the monsters. It is: "garbage in, garbage out." It is a variation of the Biblical injunction, "As ye sow so shall ye reap." Managers of large data-processing centers have learned the hard way that garbage, like crime, does not pay. So the data input process is carefully controlled, and legions of people are hired.

Sometimes, even in large installations, it is not clear whether or not the computer solution is cost-effective. But usually the argument can be made that "useful information for management" can be extracted from all those transactions and anyway, "all of our competitors have computers." Clearly, many large organizations have operated successfully without any computers whatsoever. After all, the age of data processing didn't begin until about 1960.

But let's return to our home accounting system. Like the

EVE 'N' PARITY



"He has a sweet job in process control at Hershey's."

Cartoon by Korkie

big companies, each of us has income and bills to pay. Suppose our fancy home computer is even equipped to print the checks we write. For the moment we will overlook the fact that no Commodore or Radio Shack home computer yet has a printer as standard equipment. We will also overlook the fact that computer printable continuous-form checks are not exactly what your bank sends you to write on. Still, to use a home computer, every day or once a month, we will have to sit down, key in data, and proofread it (or verify it in some other way). The time consumed in doing this is probably as much or more than simply writing out the checks directly.

Furthermore, a good deal of other work has been introduced. There are programs to be written (or upgraded), and the actual running off of the month's checks and reports takes time too. No good data-processing center operates without paper and machine readable "back-up" copies of its files. What if the machine is in the shop being fixed or that nifty, el cheapo floppy disk store crashes and scratches all the data off track 53 or your master file disk? Back-up is essential to protect against this sort of problem.

And what about the cost of the computer operation itself? Capital outlay, materials cost, repairs, and maintenance. All of these costs just to use a computer to do what will take less time by hand? With what benefits? The kind of "management information" the average person needs is practically trivial. Is the bank account getting low? (Cut out the wife's beer.) Is the car costing too much to operate? (Get a horse.) And so on. Do we really need home computers to find these things out? I doubt it. The only excuse for using a computer this way would be to say that you do home accounting on a small computer as a "hobby." It might be amusing for a while but I suggest that even hobbyists will find more meaningful ways of using their computers.

Those of you with long memories and early subscriptions, may remember that in the September issue, I told a sad tale about a colleague of mine who soon after suffering at the hands of one programmer put himself into the hands of another. In the column, I was skeptical as to whether round two would be any better than round one, especially since the new programmer chose to start over again rather than to repair programmer one's disasters.

There is a nice postscript to the story. Much to my surprise and, I might add, pleasure, the new programmer has put the project back on the track. More importantly, he has completed a successful text-editing program on schedule. Which proves the point of the column. As I said in the last line, "Nothing is really impossible, even if you have to start over again."

Those of you with even longer memories and charter subscriptions may also remember that in the first number of this column, I proposed to review actual systems. I also mumbled something about a "disaster-of-the-month" award. Well, I still intend to get around to reviewing systems, but so far, nothing I've seen has really aroused my wrath or beatitude, either of which is a prerequisite to a spicy review. The "disaster-of-the-month" award is still awaiting a few good disasters. Anybody out there have any good suggestions? ▼

How Well Are You EATING?

Are you curious about the nutritive content of your favorite budget casserole? Wondering if that new recipe will fit into your special diet? Or just want to know if that delicious dessert had any "redeeming" nutritional value? Team up your computer with NUTRIVALUE* and get answers!

The NUTRIVALUE personal nutrition analysis programs allow you to analyze recipes, meal plans, and daily or weekly menus on your home computer. Just type in the list of ingredients; your computer, running NUTRIVALUE software, will compute and print (or display) the analysis. NUTRIVALUE comes in two versions; pick the one that suits your configuration:

NUTRIVALUE I

- Analyzes 12 nutrients
- Numerically coded input
- Contains nutrient data for 53 food items; user expandable
- Tabular output format
- Written in BASIC; does not require string functions or file manipulation
- Source program requires 5K bytes of memory; user reducible

NUTRIVALUE II

- Analyzes 17 nutrients
- Ingredient specifications can be entered by name
- Choice of 100-food item or 200-food item data base
- Tabular output format
- Written in BASIC; requires string functions and file manipulation
- Source program requires 5K bytes of memory, user reducible, and file-structured secondary storage

* NUTRIVALUE is a trademark of Consultus

For more detailed information about
the NUTRIVALUE programs,
send this coupon to:

Consultus
P. O. Box 86
Arlington, MA 02174

Please send me more information about NUTRIVALUE

Name __________

Address __________

City _____ State _____ Zip _____

Computer configuration __________

ROM

COMPUTER APPLICATIONS FOR LIVING

Synthetic Skin for Your Robot and How to Make It

by Stephen J. Chiodo

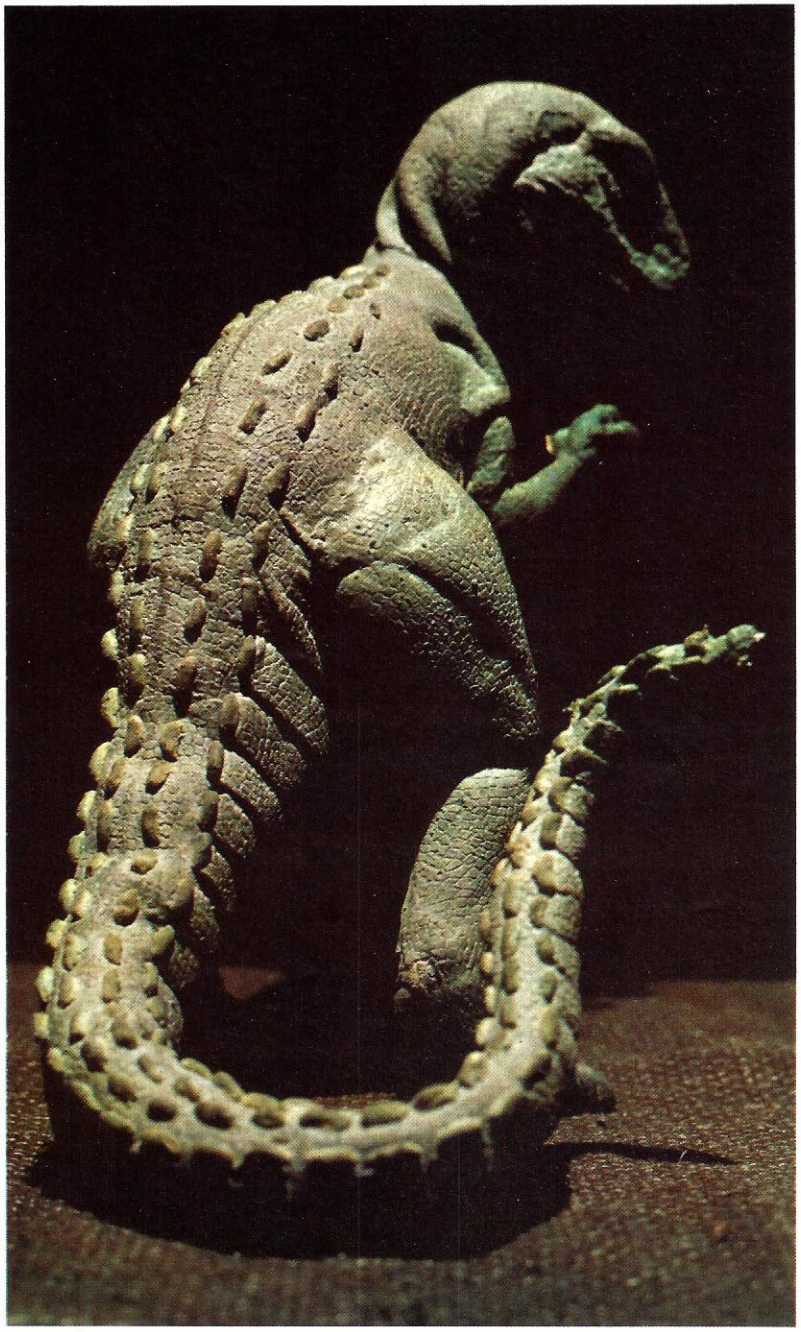
The robot of the near future will not be your metallic C3PO; instead it will be soft and genuinely human, clad in pliant flesh. What makes the Bionic Woman's skin so soft and smooth? Chrome steel? No, a realistic foam rubber skin over her bionic structure. And you can make a skin like that for your robot, be it a rudimentary turtle or a complete android that's on the drawing board.

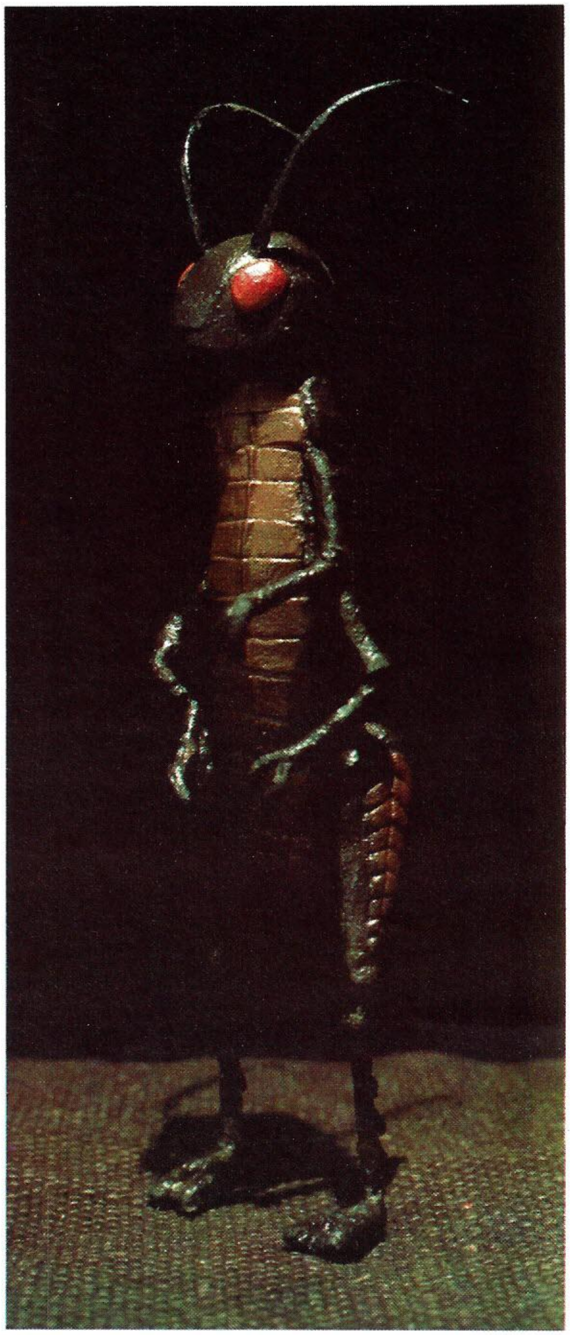
As a fabricator of three-dimensional animation models, I've developed some of the techniques you'll need to use. One of the challenges of my work, in fact, has been to construct a form capable of moving in increments as small as one millimeter. Such a model must be able to hold each position for the single frame exposure needed in animation. Along with the construction of an armature or skeletal structure, one must create a lifelike body bulk that is strong enough to withstand constant handling, yet soft enough to stretch and compress the way actual muscle and skin does.

My interest has been in the fabrication of this exterior surface, this lifelike physiognomy, be it for animals or human forms, that will do everything the real thing does, except sweat. The material best suited for these creations turns out to be a liquid foam rubber latex. The procedure for working with the foam—the preparation, casting, and finishing of a rubber model—is rudimentary enough so that with a little practice you can easily turn out your own bestiary.

What does this have to do with computers and engineering? Well, as I've advanced in my craft I've seen its applications in areas other than animation. For example, the technique can be used in the fabrication of foam rubber human appendages and, of course, even entire robots. But how to make these creations move under their own power without the aid of movie magic? That's where you and microelectronics enter the picture. With the recent advances in large-scale integration, component miniaturization, and micromechanical engineering, I can see a potential solution to the real time movement of inanimate objects. Intelligent robots are, of course, much further away. In the meantime, here is my technique. With it I look forward

Photographs courtesy of the author



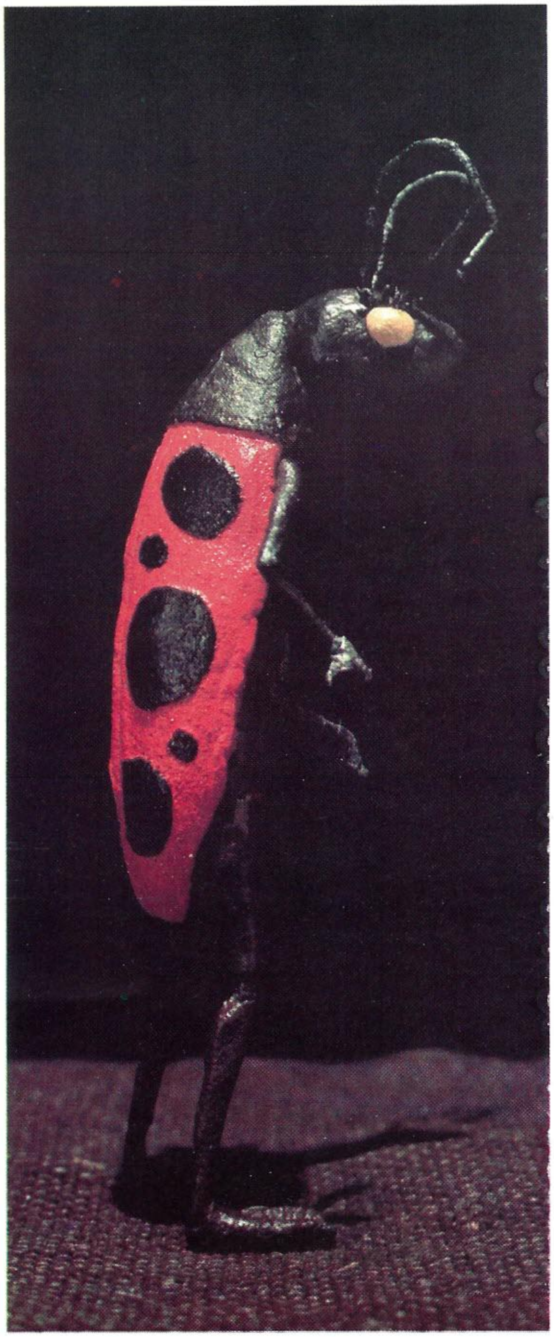


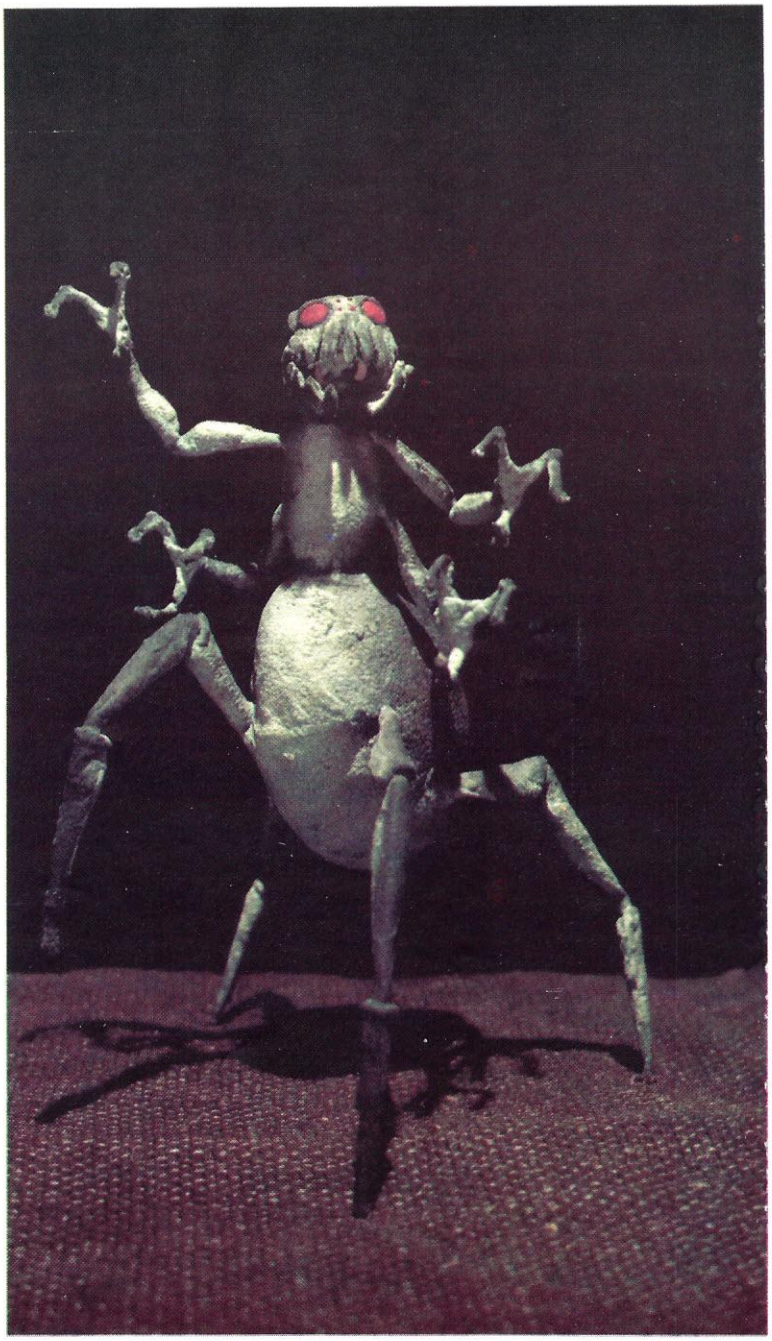
to the possibilities of our collaboration. I'll supply the form, you supply the movement, and perhaps eventually some software gnome will come along and touch our creation with a magic wand.

As a model to aid in the explanation of the foam process let's assume we wish to construct the hand and forearm of a man. The internal structure is the hardest part. It must be made to exact specifications and proportions to match the human form. The mechanics should conform to the already-defined movement of the joints.

The body bulk and exterior surface of the hand and forearm will be molded in an oil-base plasticine clay. This clay is very easy to work with—it holds detail and can be reworked at any time. To begin with, the internal mechanism is covered with tinfoil or plastic wrap, because the clay is molded directly on the armature.

Sculpting on the armature will insure proper placement of the armature within the clay piece (which will later be foam) so that it will be able to move

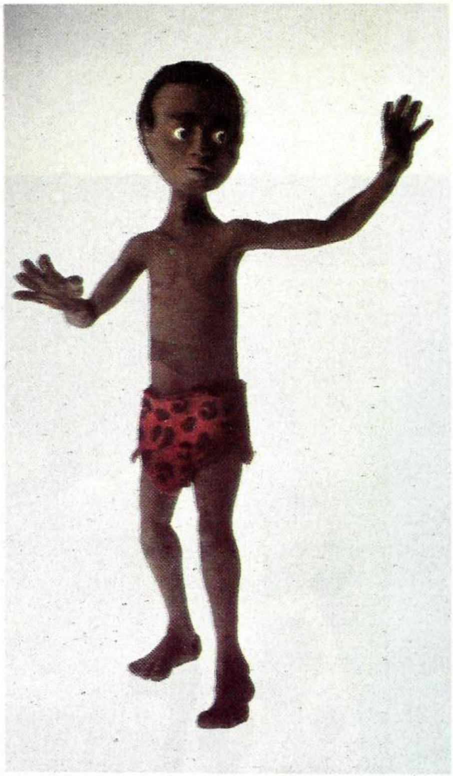


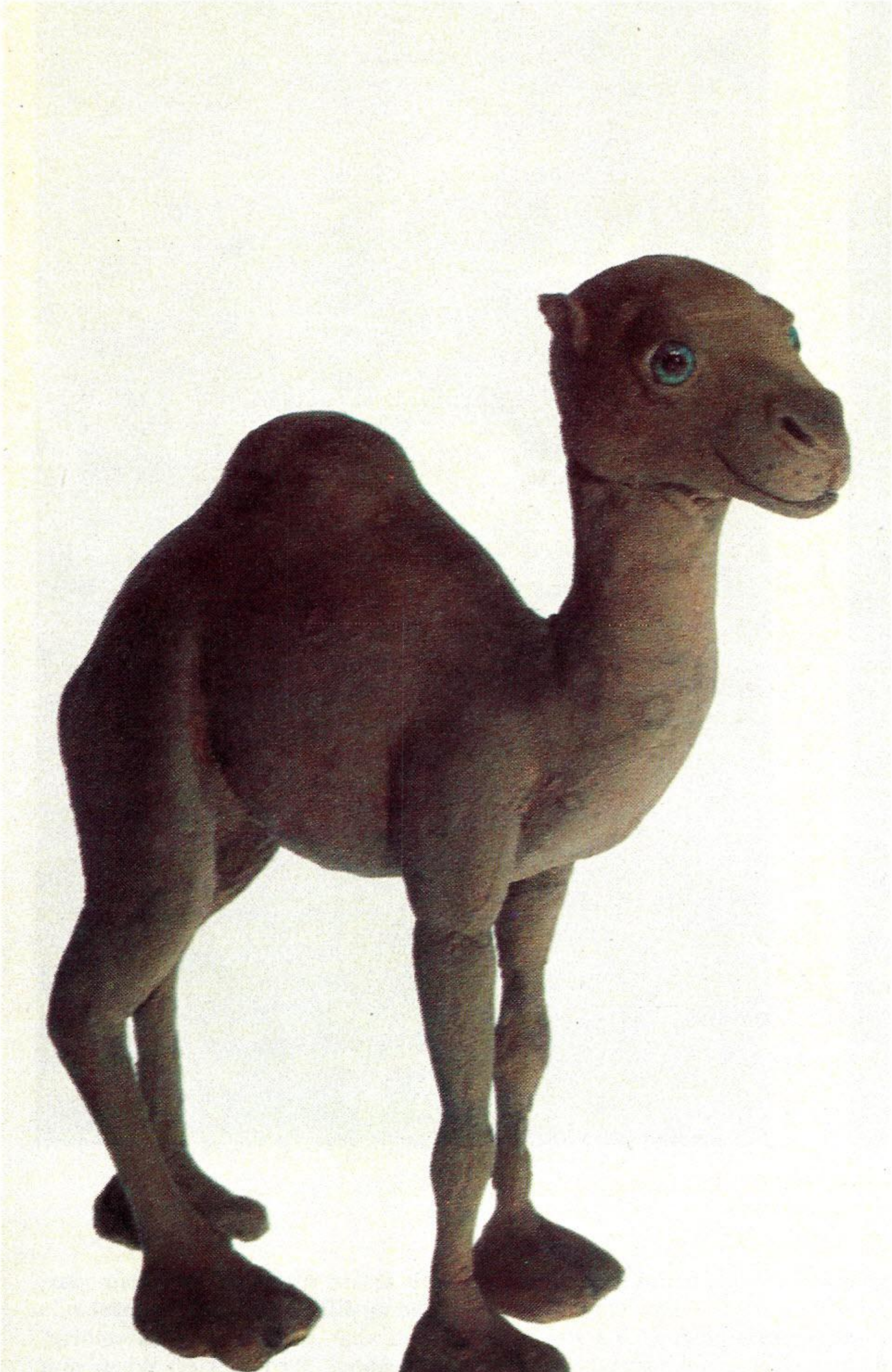


and function as planned. Using the armature as a reference to the skeletal system of the arm, you now begin to build clay pieces over it, muscle by muscle.

You should sculpt all the ligaments, bones, veins, and muscles in clay so that they will function and react with each other as the human parts do in a real arm. This is very important. The external form you sculpt must be anatomically correct. The clay muscles should be sculpted in such a way as to anticipate the stretching and compressing they will do when moved. If you allow for this in your clay piece, then when it is cast in foam it will stretch and compress very much like a real muscle.

Next, the clay muscles are covered with a skin surface. With your clay, build up different thicknesses of skin in places that require some stretching, for instance, the knuckles. When your finger is extended, an excess of skin accumulates over the knuckles and wrinkles. As the finger is bent, the skin is pulled across the joint and stretched almost smooth. Allow for this sort of displacement of flesh by building up



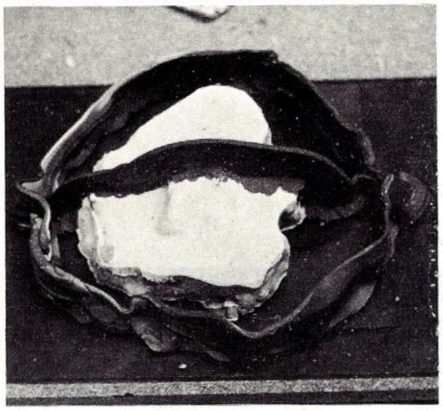


more clay at these joints, adding wrinkles so it will retract and crease in a realistic manner.

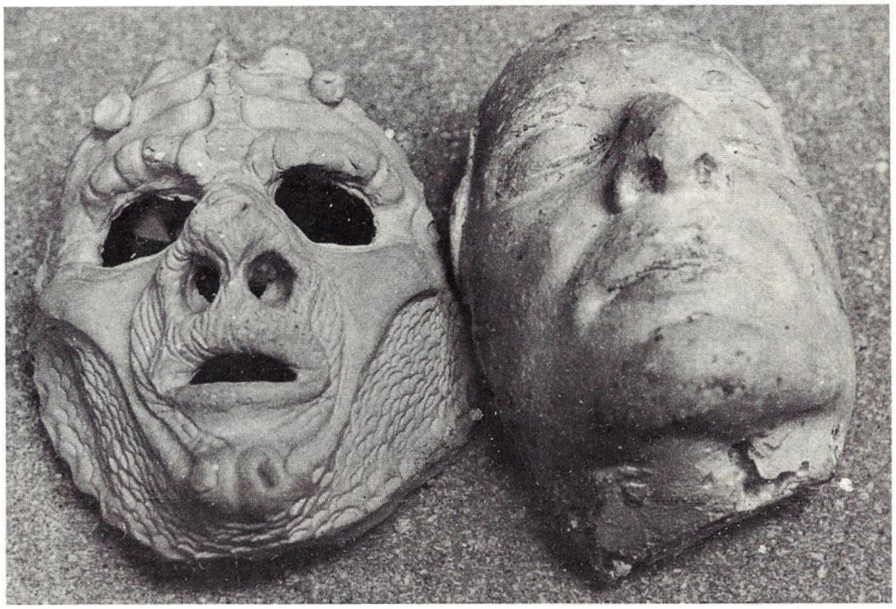
Over this skin surface you must add some skin texture. Remember that human skin, besides having wrinkles, has very subtle texture differences on different parts of the body. Such light texture differences may be added by impressing various objects, anything from cloth to the skin of a grapefruit, gently onto the clay surface.

When skin texture has been added, your clay piece is finished. It may shock you and make you feel a little uneasy because it is so lifelike. Show it to your friends and make them faint.

If time is of the essence or you do not feel confident enough as a sculptor to create a lifelike human form, there are materials on the market for making molds from living subjects. A material I've used occasionally is Silastic 382, a medical-grade elastomer produced by



A dam structure for retaining RTV as it is poured over a clay model. First RTV is poured into one side. Then the center strip is removed, the edge of hardened RTV it touched is coated with Vaseline, and the other side is poured. The Vaseline keeps the two halves from joining.



To the right is a plaster positive of a person's face. At the left is a mask, complete with desired details, made from the plaster positive. As a foam casting, it is flexible enough to be a life-like cover for the "face" of a robot.

Dow Chemical for the making of molds from live subjects. From such a mold you can cast a plaster positive and now be at the same point in the process as the guy who sculpted his form in clay.

Now we must make a mold of the clay piece (or a plaster positive dupe if you went Silastic) in which to cast the foam rubber. In the past I've used plaster and dental stone for making the mold at this stage. Since this mold will have the liquid foam rubber injected inside and will be placed in an oven and baked, I've found, however, that the plaster molds don't hold up too well. They begin to crack and thus lose their fine detail.

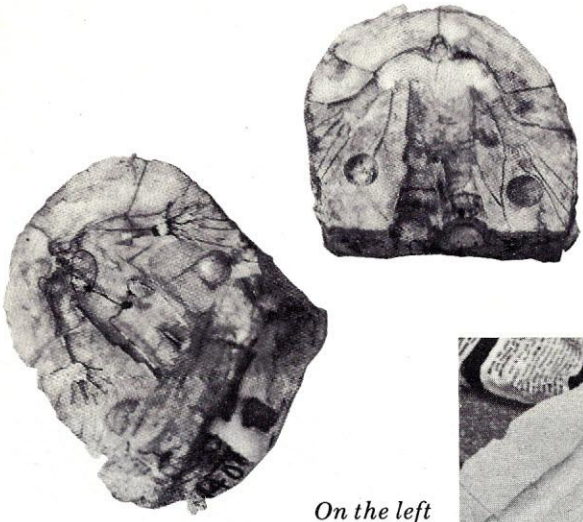
In my search for a new, more durable mold-making material I discovered a silicone rubber compound produced by GE, called RTV-11. It's a two-component silicone rubber originally intended for making highly detailed industrial molds (its elongation properties and its thermal and tensile strengths being greater than our requirements). However, it is perfectly suited for our needs as a flexible mold material for the foam process.

First, coat the clay original with a spray varnish or plastic sealer to prevent the clay from adhering to the RTV mold. Then make a two-piece mold of the arm by coating half the arm (bisecting it symmetrically) at a time with the liquid RTV compound. When it is completely cured (this will take twelve to twenty-four hours according to amount of catalyst used), coat the inside of the first half of the mold with Vaseline (to prevent the mold halves from bonding), and pour RTV over the remaining exposed section of the clay original.

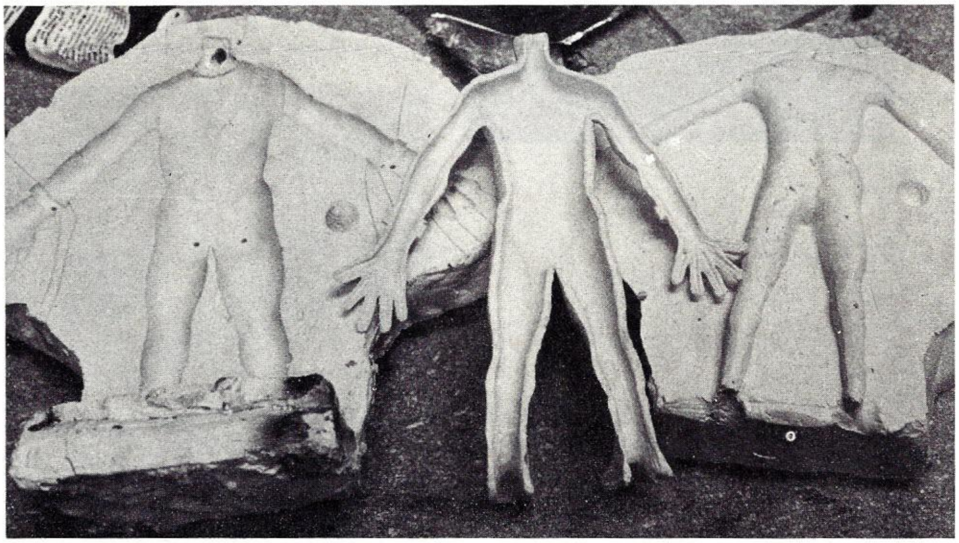
When the mold is dry, pry the halves apart and remove the clay piece. Clean out any clay that may have adhered to the RTV surface; any leftover particle will ruin the surface of the foam cast.

At this point, vent the mold with holes at the extremities (for instance, the finger tips) or wherever trapped air could collect during the injection of the wet foam. Such air pockets would prevent the foam from getting into these points; venting gives the air an escape route. Drill a pour hole wherever you feel it won't be noticed on the finished piece, as a path for the liquid foam. At this point the mold is complete and ready to produce as many duplicates as you like.

The foam-rubber compound used in the final casting is basically a liquid latex whipped at high speed with three ingredients: sulfur, zinc oxide, and a jelling agent. When the solution is injected into a mold with a grease gun or caulking gun and baked for thirty minutes at 350 degrees, the rubber cures so the air suspended within produces a foam



A two-piece RTV mold. A wire armature is registered inside one half (see lower left). Foam is injected around the suspended armature—in the case of a robot, the mechanical muscles—after the halves of the mold are closed.



On the left is the concave half of an RTV mold. On the right is the convex half, which fits inside. A thin, flexible skin (see the one in the middle, between the halves) is cast by injecting foam into the space between the molds. In this case the armature or mechanical movement is inserted into the finished skin.

rubber. A number of rubber companies distribute a foam-rubber kit; each kit has its own recipe. The compound I use is furnished by Uniroyal and costs six dollars per quart.

There are two techniques for casting a foam piece with an armature inside. According to the first, you inject the wet foam around the armature, which is secured in the mold (via registration pins suspending it inside), curing the foam around it. By the second technique, you cast the foam piece in its two separate halves, later securing the armature inside the already cured sections.

I've had the most success with the first technique. Using it, there was little adverse effect on the armature such as rust due to the moisture. However, this was never with the inclusion of a functioning electronic device. If the fabricator felt that the armature could not withstand the process, then the second method might be appropriate.

The second method is most easily accomplished by placing a sheet of tinfoil between the two half sections of mold and injecting foam into each so as to cast each half separately. When cured, it will be a simple matter to cut out the center bulk of the piece and insert the armature inside. Then you can align the sections and join them with a sealer of rubber latex. Both methods deliver similar quality, so choose the one that suits your needs.

After being removed from the mold, the foam piece usually requires a little trimming and cleaning up. There will be a slight seam running along the edge where the two-piece mold joined. With a small manicure scissors one can usually trim this down to the surface, making it virtually unnoticeable. Any pimples, due to air pockets in the mold, can be cut off at the surface and any holes due to air pockets in the wet foam can be filled in with rubber latex.

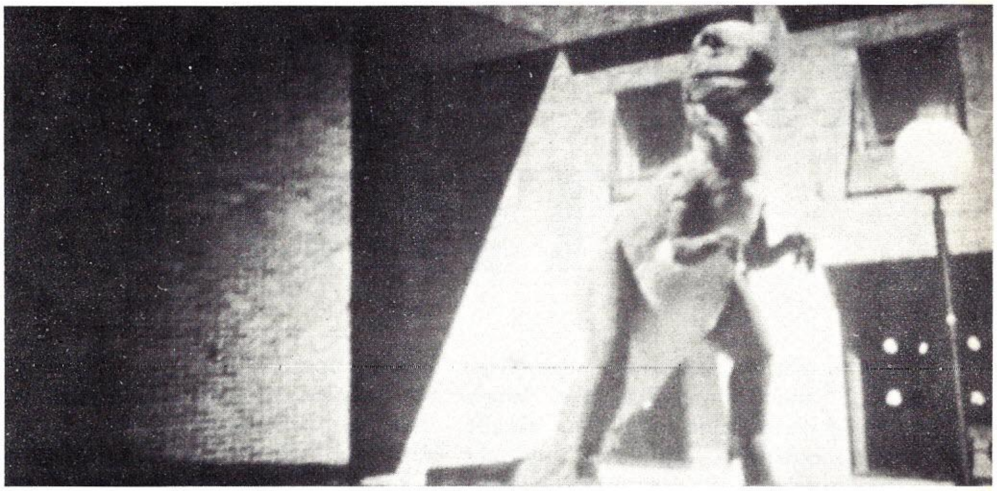
The painting of the foam piece is another critical step. The subtlety of hue and the value and intensity of skin tone are difficult to reproduce. In humans they come from many almost-translucent layers of skin, with constantly changing body fluids adding color. In Caucasians, the skin tone is made up of layers of skin tissue which are yellow with red capillaries and blue veins. A lifelike skin tone can therefore be achieved only by superimposing these colors on each other.

The paint used to color the foam piece is a mixture of dry pigments suspended in a solution of rubber latex and ammonia. This solution, when applied to the foam, penetrates the surface and becomes an integral part of the foam piece. When the pigment is applied properly, it leaves no texture, allowing the sculpted skin texture to show through.

The pigment should be applied with a soft camel-hair brush and dried with the heat of a hair dryer. There are many layers of color to apply, so quick drying helps.

Detail such as hair can be added after painting the surface. Crepe hair from a theatrical supply company works nicely and can be adhered to the foam with rubber latex.

Amazing work has been done with foam rubber and I believe it has real potential in the area I've described. If the advances of the computer and electronic fields are at the right point, those metallic robots of science fiction need never become reality. We can fill the world instead with more humanoid robots. ▼



Special effects.

The final touches—with special paint made from an ammonia base, liquid rubber, and dried pigment.

Foambots produce startlingly real animation when placed in the appropriate setting. Here the hero of the film Cricket *walks dejectedly through the woods.*

THE CODE THAT CAN'T BE

Communication is a problem, and it always has been. When it's possible at all, it's usually slow, expensive, unreliable, and not really private. Or there's some other drawback.

With phones, it always seems that you want to call just when the rates are highest, or you call station-to-station and the operator says your party is unavailable. Or you do call person-to-person so that you can leave a message if necessary and you've wasted your money: they *are* there. Privacy on the phone is also a problem—you're about to tell the party some juicy bit of (confidential) information...when your boss walks in.

Communication by mail has its hairy side as well. Everyone knows the story about the letter that took seventeen (or was it twenty-seven?) years to get across town. We once even received a bill from the newspaper with a notice stamped on the front of the wrinkled envelope: "This letter has been recovered from the waters of Long Island Sound." (It was only a few months late.) But if you want a document transmitted, it's either the U.S. Postal Service or a commercial courier.

Then there's Western Union, which combines most of those problems in one format—the telegram. (I don't even want to mention Mailgrams.)

By this time, all of you should be thinking that the field is ripe for computerization. You're right, it is a good idea, except that it's already been done. For instance, there's the Telex system, which lets your Teletype (or perhaps your computer) transmit a message to someone else's. That sort of service is expensive, too—closer to the cost of a telephone than a letter. Okay, what does that leave us with?

My approach would be to create a computerized "mailbox" system, such as those commonly in use on time-sharing computers. The idea is that you'd dial up the host computer, have your computer send a message (this is so it can be done as quickly as possible), and instruct the host computer to store the message for its intended recipient, say, Uncle Frank.

Then old Frank dials up the system, hooks his computer up to the phone, and in under a minute (something like fifty seconds for a page at 300 baud) he's got his mail. If you do all this after 11:00, the cost is competitive with that of regular mail (except for the hardware, which we'll assume the local computer freaks have donated). And the computer message can be more secure—private—than either mail or a telegram.

It can be so secure that the letters written can even be legal documents. In fact, it's possible to put an electronic "signature" on your message which will prove who the sender was! And to insure that the letter will be coded correctly—for the way to send a secure letter is in code—you tell your correspondents how to code it. Finally, just to make things fun, the code could, conceivably, be broken by someone who really wants to. But you're probably still pretty safe: they'd have to want to a lot. It would take a computer about four million years, working full time, to crack the code.

To make it positively forgeproof, just sign *your message.*

If you're inquisitive, your first two questions are probably, "How do I know when a letter has come in on this system?" and "How does such an incredible code work?"

To your first question I must answer that I'm somewhat stumped. But there are possibilities. The computer could be fitted with an automatic dialer and a voice-synthesizer, for instance, and it could make a call to you. But if we want to keep it competitive in price with mail, we'd have to avoid needless calls. You don't want to spend fifty cents just checking to see that your box was empty, either. Perhaps, instead, you could call the computer person-to-person for yourself. Someone is bound to think of a way to do it cheaply. Any suggestions?

The second question, about how the code works, is more easily answered. The answer should appeal to the budding computer freaks among us, too: this new code has its roots in, of all places, the study of computer science. In fact, the only way we know that this code is uncrackable is by figuring out how long the fastest computer program for solving it would take. The code is the brain child of two electrical engineers at Stanford University, Whitfield Diffie and Martin E. Hellman. Like many good ideas, it is remarkably simple—it would be an ideal project to implement on a personal computer.

If you've ever tried to write a BASIC program to implement a simple substitution—Caesar's (according to Martin Gardner, really Julius himself) cipher—you know it's easy to do. It works like a Cap'n Crunch Code Ring, if you've ever had one of them. What a Secret Code Ring does is shift the letters of the alphabet around: you write your message down, look up the replacement letters on your Secret Code Ring, and re-write the message with the letters replaced by the ones that are now opposite the original ones on the code ring. For instance, if the *A* on the code ring is set opposite the *G,* the word *computer* gets coded as *iusvazkx*. But as anyone who's ever tried the cryptogram in the Sunday paper knows, it's relatively easy to de-

CRACKED

by Eben Ostby

Microcrunch
the Breakfast of coders
FREE secret coder in every box!!

cipher one of these. Nevertheless, the Cap'n Crunch Secret Decoder Ring is the starting place for a really unbreakable code.

Suppose you replace the inner disk on the code ring with one with numbers printed on it instead of letters. To make things simple, you could even line the *1* up with the *A,* the *2* with the *B,* and so on. So if you encoded the word *computer,* you'd come up with *3 15 13 16 21 20 5 18.* Now turn all the numbers into two-digit numbers by putting in a leading zero where necessary, string them all together, and you've got a (large) integer:

0315131621200518

which represents the word. This process could be applied to an entire message. On a computer using the ASCII code, by the way, converting letters to numbers is no problem at all—all you do is mask out the first three bits of the byte that represents each character. There are dozens of ways you can pack the results together to save space, too, although it's not essential.

The crux of the new code scheme, however, is not this transformation but a very tricky modification. Known as a *trap-door one-way function,* it is a simple mathematical transformation that has these properties:

1. It will change any number, such as one representing a message, into another number. If M is the message and f is the function, it will produce some number C. In mathematical terms, $f(\mathrm{M}) = \mathrm{C}$.

2. There is an *inverse function*—call it g—that transforms C back into M. That is to say, there exists a g such that $g(\mathrm{C}) = \mathrm{M}$.

3. It's possible to evaluate f and g on a computer.

4. Even if you know f, it's essentially impossible to figure out what g is.

Here's what all that means. The first property just says that f is some code. The second property says that something that you've encoded using f can be decoded again. The third property—well, that's why you're reading *ROM.* But the last property is the important one. It's the trap door. What it means is that you can tell anyone how to encode a message to you. You could plaster it on the back page of *The New York Times,* or (better yet) put it in a public data bank on a computer system. But anyone who sees the method would be unable to figure out how to decode the message—he'd just know how to encode it.

With most codes—for instance, the Cap'n Crunch Secret Code Ring, also known as Caesar's Cipher, if you tell someone how to encode a message, you've also given away how to decode it. For this reason, you have to be discreet about the process. But with the trap-door function, you can be completely irresponsible! What's really important, though, is that anyone can send you a message.

Stanford's Diffie and Hellman did not come up with any ideas for what function actually has all these properties in their paper on the subject. However, a group of computer scientists at M.I.T.—R. L. Rivest, A. Shamir, and L. Adleman—have come up with a brilliant idea.

Based on an elementary theorem in number theory (Fermat's theorem), their system just involves raising the number that represents the message to a power, dividing that by another number, and taking the remainder. The remainder is the encoded message!

Illustrated by Linda Smythe

Try a simple-minded example to see what this means. (Remember, the numbers in this example, for simplicity's sake, are *much* smaller than the numbers actually used in the code, as will be clear below.) Suppose your message is represented by the number 5. Raise that to a power, let's say 5^3. $5^3 = 125$. Now divide that by another number, say 11. 125 divided by 11 equals 11, remainder 4. So the remainder, 4, is the encoded message.

When you wish to decode the message, you do the same thing—except that you use a different number as the power you raise the message to. Obviously, the numbers you choose are crucial to the method. Mathematically, the method looks like this:

$$C = f(M) = M^e \pmod n$$

The notation (mod n) just means that you divide the result of M^e by *n* and take the remainder. Not just any *e* and *n* will work, of course. Specifically, *n* must be the product of two large prime numbers—call them *p* and *q*. The reason that they must be large is so that *n* will be doubly large. For instance, if *p* and *q* each have twenty digits, *n* will have about forty digits. The larger *n* is, the harder it is to break the code. The reason it's harder is that the only way to break the code is to factor *n*—in other words, figure out what numbers multiplied together will produce *n*. For an *n* with fifty digits, it can take as few as four hours to figure this out on a fast computer. However, if you double the number of digits in *n*, so it's a hundred-digit-long number, it would take seventy-three *years* to compute the divisors *p* and *q!* If *n* is two hundred digits long, it would take a fast computer, working 'round the clock, *3.8 million years* to figure it out. Got the idea?

As I said before, decoding the message is the same process as encoding it—except that a different power is used. I'll call it *d* (for "decode"):

$$M = g(C) = C^d \pmod n$$

What values do you use for *e* and *d?* The only restriction is that the remainder of their product must, when divided by $(p-1)*(q-1)$, equal 1. That is,

$$e*d = 1(\mathrm{mod}[p-1][q-1]).$$

Now you know everything that there is to know about encoding using a trap-door function. Presumably, to use the method, there would be a public table of values for *e* and *n* (but not *p, q,* or *d,* because knowing these is tantamount to breaking the code). Each user of the system would supply his own values. When you wanted to send a message, you'd just look up (or have your computer look up) the values for whoever was to receive your message, and encode the message according to those numbers.

This system has another trick, though—you can *sign* a message and make it positively forge-proof! This is how that works: instead of just encoding a message once, using the number in the table, someone wishing to sign a message would encode it twice. For instance, if you wanted to sign a message to Uncle Frank, you'd first encode the message using your secret *decoding* numbers. This is important—you can "decode" a message that has not yet been encoded, and since encoding and decoding are inverse processes, the result can be "encoded" later to reveal the original message. The difference between first decoding, then encoding, and doing it in the normal order is just this: anyone can decode the message now, since your encoding numbers are public property. But *only you* could have encoded the message, since only you know the secret numbers that are used for decoding your incoming messages—the numbers you just used for encoding the message. Got that?

Now you just encode the message that you encoded using your decoding numbers, but this time using Uncle Frank's publicly known encoding numbers. When Frank receives the message, he'll first decode it using his secret numbers. Presumably, he knows that it was you who sent it; probably you signed it in English with your name. So he "encodes" it with your public encoding numbers, and the result is the message that you alone could have written!

The whole system is quite tricky—but at the same time wonderfully simple. In fact, it's so simple that it's just begging to be implemented on a personal computer. Perhaps, when it is complete, you can use it to send fan mail—to *ROM.*

Program

So you want to write a program to encrypt and decrypt messages, do you? The process is fairly simple—but beware! As with all (most?) good things, there's much more below the surface than you'd guess at first. To put it another way, although the operations you need to do are mathematically simple, they end up being tricky when you get down to programming. Of course, that's what really makes them fun!

The basic procedure for both encrypting and decrypting (encoding and decoding) messages is simply to raise the message—the number that represents it—to a power, and divide by some other number; the *remainder* is what you want. If the message is M, the power E, and the divisor N, you want to find $M^E \pmod N$. In BASIC, this amounts to the statement:

```
C=MOD((M**E),N)
```

if the system has the MOD function; otherwise, it's the following sequence:

```
C=M**E/N
C=N*(C-INT(C))
```

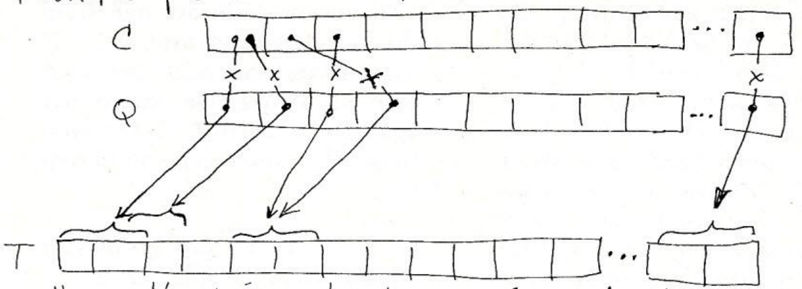
These two statements compute, first, M^E, then divide that by N. Then the remainder is found by multiplying N times the fractional part of the first computation. In APL, the same result is found by saying:

```
C ←N|M*E
```

But there's a catch. You can't represent most messages in one APL or BASIC variable. Suppose, for instance, that your computer uses a four-byte word. Well, the old Bowmar Brain tells me that the largest number which that will hold is about 16^8, or around $4*10^9$ (four million). How many characters can you squeeze into that word? Well, if your alphabet is 37 characters long (26 letters, 10 digits, and one space), you can fit up to $\log_{37}(4*10^9)$, or six characters. If you have an alphabet of 256 characters, which is what you get if you use one byte per character, you get exactly four characters per word. You see, you can't fit much of a message in four or six characters. So you need a bigger word size. This is true regardless of whether you know anything about how I came up

the Code!

Multiple-precision multiplication:



Here I'm trying to show a few of the separate multiplications that occur in step 5 of the algorithm, and what part of T they get added to. The binomial theorem tells how many additions take place, if you want to figure it out.

(This is merely an illustration—it is not necessarily complete or correct.)

with that $\log_{37}$ back there. There are some more reasons, too.

You will recall that it's suggested that you use a 200-digit number for the divisor. This is because it's really hard to factor a 200-digit number into its components, and that's the heart of the code system. Even 100 digits will do. But if the number is only nine digits long, which is about how long a number can be which will fit in a four-byte word, then it's pretty easy to factor it in a few minutes on a fast computer. Well, you can't fit that big a number in one word on your computer, so again, you need a bigger word size. (If you're thinking that you can represent the number in scientific notation, forget it. You *need* all 100 digits.)

Okay, so now you know why it's not as simple as it would seem. Here, then, is an algorithm for doing the decoding and encoding:

1. First, you need to turn the message into a number. You know by now that it's going to be either a really big number, or else a few really big numbers. If it's more than one, remember right now that you'll follow every step in the algorithm for the first number, then come back here and do the same procedure for each of the remaining ones. But to turn the message into a number: the message already *is* a number. Since it's residing in the memory of your computer (or it will be when you get the program running), it must be represented as a number. In particular, it's probably in ASCII code, where one byte, which can hold any number between 0 and 256, represents one character. I know you don't have 256 different characters to choose from, but it is a handy way of storing characters. So your "large number" is really a *string* of bytes. Since the computer operates on one byte at a time (or maybe two or four, but not the fifty or so that your message takes up), what you have to do is the raising-to-a-power and dividing on a massive scale. That follows.

2. Right now, I will assume you've picked out a number to use as the power to which you raise M, the message. This power I'll call E. E will probably take up more than one byte, too, but not necessarily more than one word (four bytes). Since E is in the computer's memory, it must be stored as a binary number. More about that later; I just want you to keep it in mind.

3. Also sitting in memory you've presumably got a variable C, which is where the result is going to end up. Likewise, you've got to have a variable N, which is the divisor. It's easiest to accomplish everything if M, C, and N are all the same size. This is probably around fifty bytes for typical messages. C must be set to the value 1. For a fifty-byte-long number, this just means that the first forty-nine bytes are all zero, and the last one contains a one. Representing it as a long binary number, it looks like this:

0000000000...00000000001

N you must set to the value of the divisor. This is a long number which I'll mention again later on.

4. Now you need another variable. This is a small number, call it I. I should be set to the number of bits in E, minus one. In other words, your program should count down the bits in E, starting from the left-hand end. When it comes to the first non-zero bit, stop. Subtract the number of bits counted from the number of bits in the word in which E resides. For instance, if E is the number 6, and sits in a one-byte-long space, you'd count from the left 6 places before you found a one-bit:

```
   E: 0 0 0 0 0 1 1 0
Count: 1 2 3 4 5 6
```

Subtract this from 8, and you get 2, which is the number you want for I. (In practice, E will be considerably bigger than 6, but the idea is the same.)

5. Now you have to square C, divide by N, and take the remainder. To square C, you have to multiply it by itself. I'll assume your computer can multiply, or you've got a subroutine that will do multiplications for you. You need a variable that's twice as long as C is (maybe 100 bytes? or if C is not exactly 50 bytes, create a space that's twice as long). Set the whole thing (call it T) to zero. You need two variables that are, say, one word or smaller each. Call them J and K. Set them both to the number of bytes that C takes up.

5a. Check if $J=0$. If it does, set $K=K-1$, and check if $K=0$. If *it* does, go to step *5d*, but if $K \neq 0$, set $J=$ (its original value) i.e., the number of bytes in C. But if J was not equal to zero, don't do anything in this step.

5b. Multiply byte J of C times byte K of C. Byte J is the Jth byte from the left-hand end of C. Another way of looking at it is that if you add the address of C to the value in J and subtract one, you've got the address of the Jth byte. Yet another way of looking at it is to say that C is in an array (called C, of course) and that the Jth byte is C(J). So get the Jth byte, get the Kth byte, and multiply them together. The result will have two bytes in it. Add these two bytes to T, but do it this way: the high-order byte (left-hand byte) of the result is to be added to the $(J+K-2)$th byte of T. The low-order byte of the result is to be added to the $(J+K-1)$th byte of T. Be sure to handle any carries properly. If adding the two bytes yields an overflow (a carry), add that bit to the next byte to the left. That, in turn, may result in a carry, so check that, too. And so on.

5c. Set $J=J-1$, and go back to step 5a.

5d. Now you've squared C. A note about this procedure: If you have one of the variables J and K—I'll say K, point "into" another variable, instead of C, you can multiply any two multiple-byte-long numbers together. What I mean is that if you've got another variable, call it Q, and you replace the sentence in step 5b that reads "So get the Jth byte, get the Kth byte, and multiply them together" with one that reads "So get the Jth byte of C, and the Kth byte of Q, and multiply them together," you've got yourself a handy algorithm for computing C*Q. You'll need this later, for multiplying C*M.

5e. Now you have to divide by N, and take the remainder. For this, create a variable (call it J again if you want) with the number of bytes that N has in it. Probably this is the same as the number of bytes in C.

5f. Check if T is less than N. Do this by finding the greatest (leftmost) one-bit in N. Now check T to find the leftmost one-bit in it. If the leftmost bit in N is greater (farther from the right-hand end of the entire number) than the bit in T, then T is less than N, so you can go to step 6.

5g. Check if $J=0$. If it does, go back to step 5e, and continue from there.

5h. Subtract the Jth byte of N from the Jth byte of T. If you must borrow to do this, be sure to borrow. Don't forget to carry the borrow as far left as you need to! (To borrow, subtract one from the next left-hand byte of T. If that makes the next left-hand byte of T negative, move left again and borrow. When you finish, then you can go on to step 5i.)

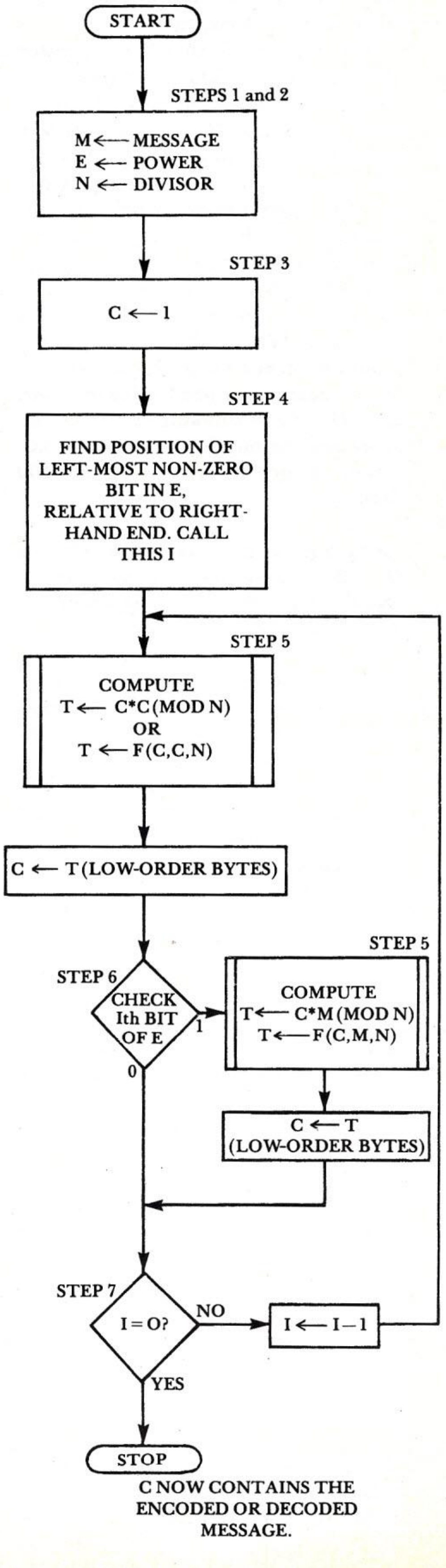
5i. Set $J=J-1$. Now go to step 5g, and continue from there.

6. Congratulations! you finished step 5! T now contains the remainder when you divide C^2 by N. Now, transfer the right-hand half of T back into the space you used for C. Next, check the Ith bit from the right of E. In this case, the right-hand bit is bit number 0. So check this bit. If it's a zero, you may go on to step 7. Otherwise, you must multiply C times M (M is the message, remember?) and divide by N, taking the remainder. This is just step 5 again. Look back at step 5d. This tells you how to do that. You'll probably want to write step 5 as a subroutine so you can call it, giving C, M, and N as parameters this time, and C, C, and N as parameters the first. If you don't know about subroutines, now is a fine time to read up. Anyway, transfer the result, which is sitting in T, back into C by again taking the right-hand half of T and moving it to C. The left-hand half, if you're worried, should be all zeros anyway—thanks to step 5f.

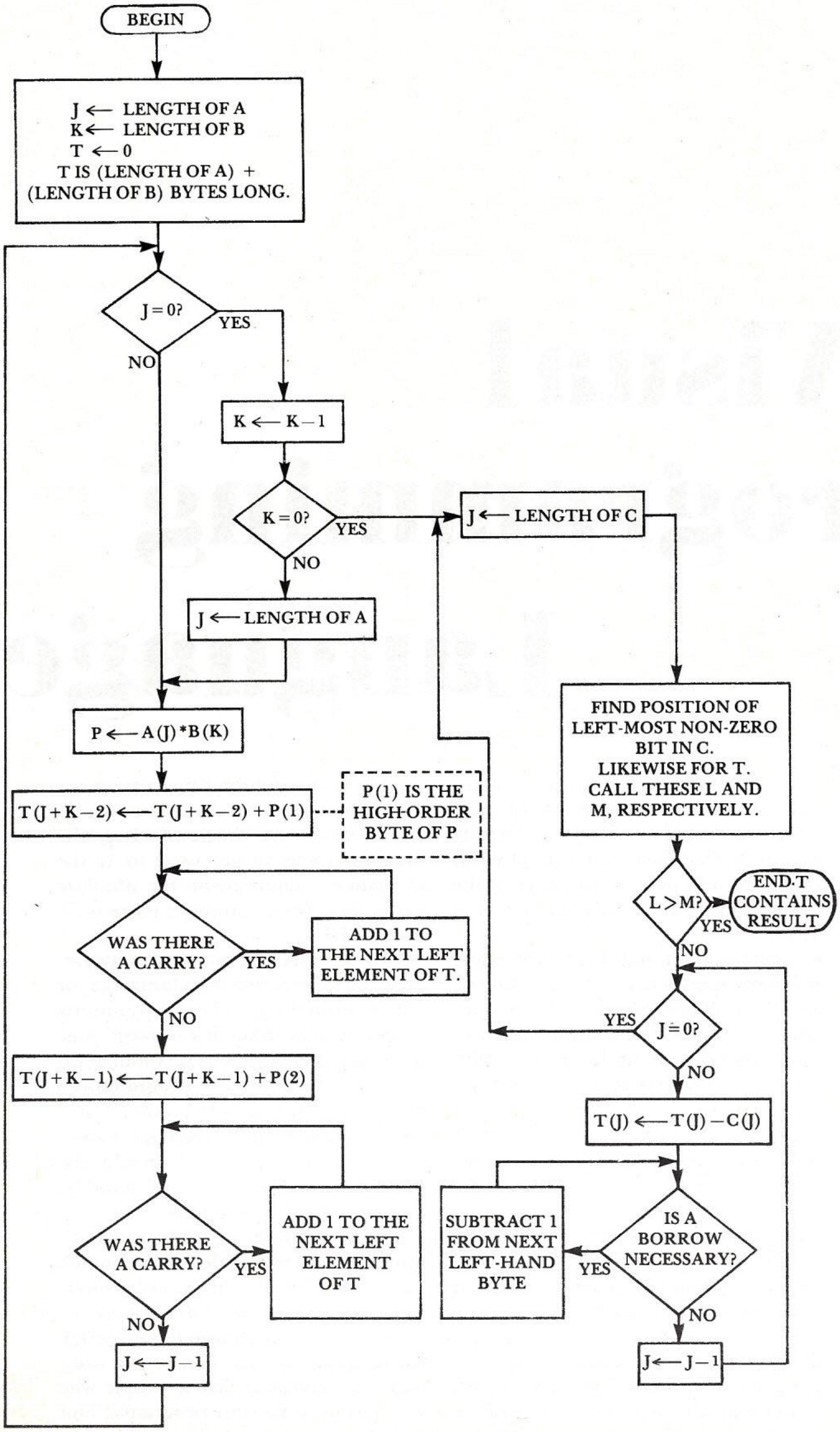
7. Now check I. If $I=0$, you are done—C contains the encoded message (if E was the encryption key) or the decoded message (if M was already in code and E was the decryption key). But if $I \neq 0$, then you've got to subtract one from it (set $I=I-1$) and go back to step 5.

The other remaining problem is picking out the encryption key numbers. First, you need to pick out two long prime numbers—preferably fifty or more digits long. Call these numbers a and b. Now the number N you use in the above algorithm is just the product of a and b. (Steps 5b—5d tell you how to do "big" multiplications.) You also have to choose two numbers E and D. First, you must choose D. D must be relatively prime to the product $(a-1)(b-1)$—that is, the numbers D and the product must not have any factors in common. Finally, you must choose E, such that $(E^*D)/((a-1)(b-1))$ leaves a remainder of

FLOWCHART FOR ALGORITHM FOR COMPUTING M^E (MOD N)


START
STEPS 1 and 2
M ← MESSAGE
E ← POWER
N ← DIVISOR
STEP 3
C ← 1
STEP 4
FIND POSITION OF LEFT-MOST NON-ZERO BIT IN E, RELATIVE TO RIGHT-HAND END. CALL THIS I
STEP 5
COMPUTE T ← C*C(MOD N) OR T ← F(C,C,N)
C ← T(LOW-ORDER BYTES)
STEP 6
CHECK Ith BIT OF E
1
0
STEP 5
COMPUTE T ← C*M(MOD N) T ← F(C,M,N)
C ← T (LOW-ORDER BYTES)
STEP 7
I = O?
NO
I ← I−1
YES
STOP
C NOW CONTAINS THE ENCODED OR DECODED MESSAGE.

FLOWCHART FOR COMPUTING
F(A,B,C) = A*B(MOD C)
FOR ANY A, B, AND C


BEGIN
J ← LENGTH OF A
K ← LENGTH OF B
T ← 0
T IS (LENGTH OF A) + (LENGTH OF B) BYTES LONG.
J = 0?
YES
NO
K ← K − 1
K = 0?
YES
NO
J ← LENGTH OF A
J ← LENGTH OF C
P ← A(J)*B(K)
T(J+K−2) ← T(J+K−2) + P(1)
P(1) IS THE HIGH-ORDER BYTE OF P
FIND POSITION OF LEFT-MOST NON-ZERO BIT IN C. LIKEWISE FOR T. CALL THESE L AND M, RESPECTIVELY.
WAS THERE A CARRY?
YES
NO
ADD 1 TO THE NEXT LEFT ELEMENT OF T.
L > M?
YES
NO
END. T CONTAINS RESULT
J = 0?
YES
NO
T(J+K−1) ← T(J+K−1) + P(2)
T(J) ← T(J) − C(J)
WAS THERE A CARRY?
YES
NO
ADD 1 TO THE NEXT LEFT ELEMENT OF T
SUBTRACT 1 FROM NEXT LEFT-HAND BYTE
IS A BORROW NECESSARY?
YES
NO
J ← J − 1
J ← J − 1

1. All of these things can be done on a computer, too.

Here are a few references if you don't have the knowledge of mathematics that's necessary. The best one you can get is the paper that describes the system. It is called *On Digital Signatures and Public-Key Cryptosystems,* by R. L. Rivest, A. Shamir, and L. Adleman, and it's technical memorandum #82 of the Laboratory for Computer Science at the Massachusetts Institute of Technology. You may obtain a copy by sending a stamped 9-by-12-inch envelope (35 cents in postage), self-addressed, to Mr. Rivest, at the above laboratory. The address is 545 Technology Square, Cambridge, MA 02139. The article will also be published in the Communications of the Associations for Computing Machinery.

Martin Gardner's *Scientific American* column, "Mathematical Games," featured this system in August, 1977. Mr. Gardner's columns are a delight to the mathematicians among us.

The system was proposed by W. Diffie and M. Hellman in the paper "New Directions in Cryptography," which appeared in *IEEE Transactions on Information Theory* in November, 1976.

A superb, if somewhat advanced, series of books on methods in programming (among other things) is D. E. Knuth's *The Art of Computer Programming.* The first two volumes of the book (especially the second) would be of use to the programmer of this system.

Finally, a book on number theory is a must if you really want to get into the system. It would be of some use in determining the numbers a, b, E, D, and N, as well as giving some insight into the elegant mathematical proof that the authors of the first paper give. Rivest, *et al,* recommend I. Niven and H. S. Zuckerman's *An Introduction to the Theory of Numbers.* There are others.

There are lots of other things I could tell you. For instance, since it's far better to write this program in assembler than in BASIC (all these byte and bit manipulations are simple in assembler, but difficult in BASIC), you ought to read up on assembler for your computer. But the best advice I can give you now is: good luck! ▼

TLC
The Visual Programming Language

I'm about to let you in on one of the best kept secrets of computer programming: it's fun. It's also fascinating—perhaps the greatest intellectual challenge ever devised by man.

If this is so, why aren't we all having fun? One reason is the unbending, intractable, literal nature of the computer hardware itself. The other is the difficulty of communication at the man-machine interface.

Personal transportation by automobile provides a good analogy to personal computing. Before cars came along, you traveled on foot or horseback, just as most people use pencil and paper or a calculator as their basic problem-solving tools. To extend your range, you rode on someone else's train or bought time on someone else's computer.

Personal computing is about to leave the world of the horseless carriage and enter the world of the Model T. As it does, the people who design computer systems are going to have to give more attention to making their systems easy to use. Until now, most of their effort has been devoted to reducing the cost and complexity of the physical hardware and pushing the state of the art in order to increase density and speed.

The result of this hardware effort *has* been spectacular. Within the last year both Radio Shack and Commodore have announced complete computer systems for under $600. (Will this seem extravagant two years from now?)

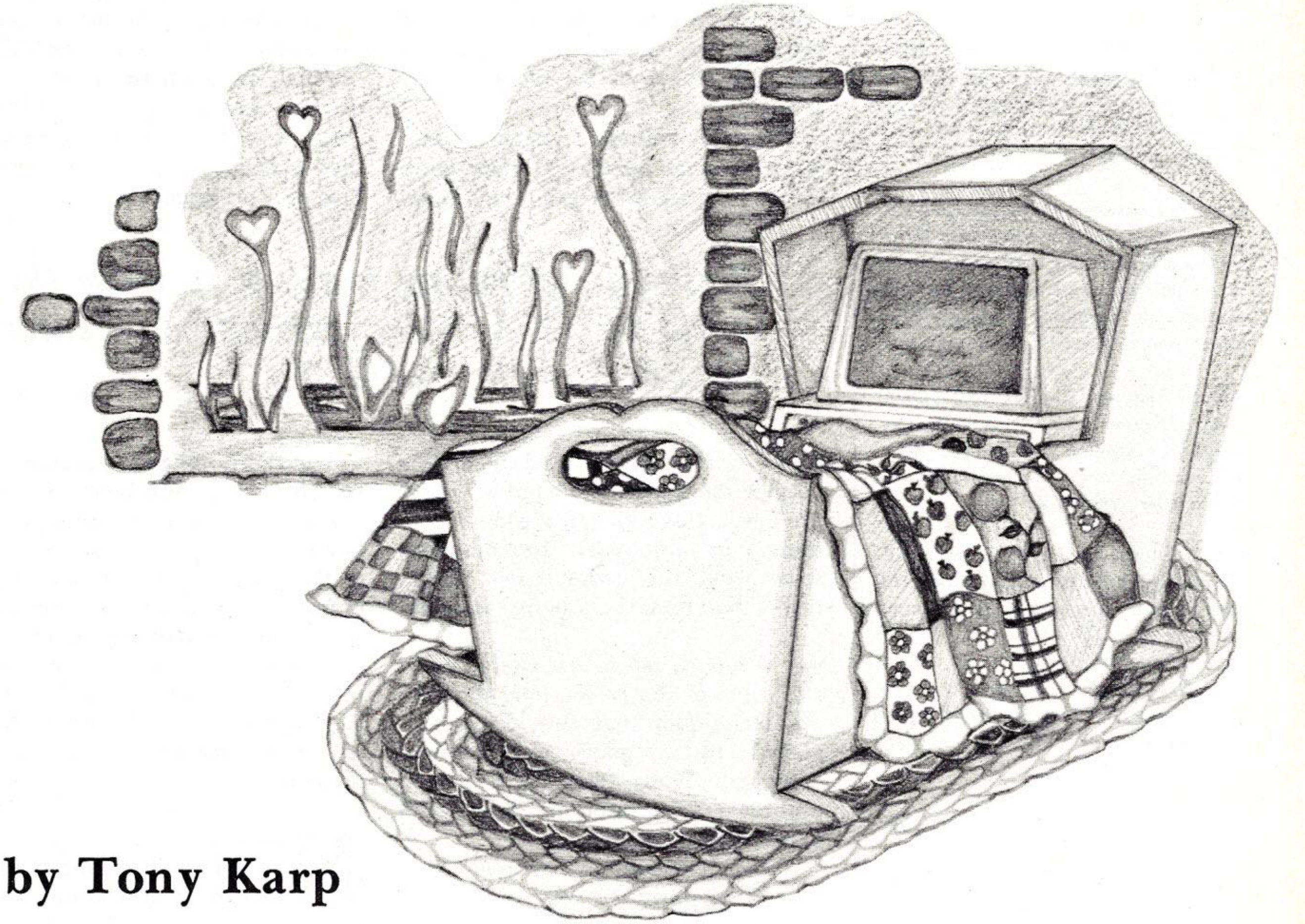
So far, so good. The hardware is now within reach of anyone who can afford a good hi-fi system. Now what? What do we do after *Star Trek, Blackjack, Bagels,* etc? Pretty soon, along comes the urge to create, to harness the power of the computer for our own uses. Balance my books? Control the energy in my house? Educate my children? Why not? That's what we bought it for. Enough of this canned software; it's time to flex our creative muscles.

The transition from being a program user to being a program creator brings a number of rude shocks. Perhaps the toughest thing to get used to is the computer's requirement for absolute perfection. For a computer there is no almost right.

Now we enter the world of software. First, we have to choose a language for our programming. The programming language is the main link between your mind and the computer. It is your common vocabulary—the programmer's most important tool. It follows that a good programming language should be easy to learn and use. It should also help you to produce correct, reliable, and well-crafted programs.

There is another aspect of programming languages that doesn't get much attention. This is the *visual* aspect. Most programming information is, after all, transmitted visually. Although the language is your link to the computer, it's also your link to people who will have to read your programs. This includes you.

Illustrated by Linda Smythe



by Tony Karp

A good programming language shouldn't stand in the way of creating and organizing large programs. It should make it simple to break the program into easy-to-handle modules that can be modified without generating disastrous side effects in other modules.

Right now, most programming applications that deal with real-world, real-time events are written in assembly language. (I will use the terms *assembly language* and *machine language* interchangeably since they both imply working with the computer's own instruction set.)

Assembly-language programs have been shown to be less reliable (more likely to contain a lurking bug) than programs written with a higher-level language. By "high-level," I mean any language where one instruction is equivalent to several machine-language instructions.

Another way of looking at this is to examine the "atoms" of the two types of languages. In assembly language, the atoms are the computer's instruction set:

```
LDA  GPAY
SUB  DTOT
STA  NPAY
```

In a high-level language, the atoms more closely resemble English:

```
NET:PAY=GROSS:PAY
        -TOTAL:
        DEDUCTIONS
```

Now you can express an idea rather than a list of machine instructions. Using a high-level language makes the computer think like you rather than vice versa.

The assembly-language programmer must know the intimate details of the particular computer he's using. Since the instruction sets of the various computers are so totally different, most programmers who work in assembly language tend to work with just one machine.

On the other hand, the programmer who works in a high-level language is insulated from these details. An ideal high-level language should be so independent that it will run on any computer. BASIC is a well known example of such a machine-independent language. A program written in BASIC should run on any machine that has a BASIC interpreter. It's interesting to note that each manufacturer tries to squeeze new features into his version of the language, thus destroying BASIC's machine independence.

If a language is truly machine-independent, software design can begin before the computer hardware is completely specified. Not only that, but software written for an older computer will run on a newer one without having to be completely rewritten.

Documentation is another part of the visual aspect of programming. Your program may be read by other programmers. Have you left a clear

ROMtutorial ROMtutorial

Software (vs. Hardware): The programs run on a computer. A computer system consists of hardware—the computer and its accessories—and software—the programs which make the hardware work the way you want it to.

Assembly language: A programming language which allows a computer user to write a program using mnemonics instead of numeric instructions. It is a low-level symbolic programming language which closely resembles machine code language.

Machine language: The internal programming language used by a computer. Different computers usually have different machine languages, which may or may not be compatible.

BASIC: Acronym for Beginner's All-purpose Symbolic Instruction Code. An easy-to-learn, easy-to-use programming language especially adapted for use with mini- and microcomputers, as well as time-sharing systems. It provides anyone using the computer with instantaneous feedback on whether he's doing all right or making a mistake.

Interpreter: A computer program that translates each source language statement into a sequence of machine instructions and then executes these machine instructions before translating the next source language statement.

Documentation: The manuals, diagrams, reference books, printouts of programs, and any other printed material supplied with a computer system.

Loops: Segments of a program which are executed a number of times. Usually they consist of a series of instructions followed by a test to see whether the program should execute the preceding instructions, or continue with the instructions that follow.

Branching: The selection of one or more possible paths in the flow of control, based on some criterion.

GOTO statement: An instruction which alters the normal order of execution of instructions in a program.

path for them to follow? Will you be able to read your own program in a year or even a month from now? If the program was written in a high-level language, the odds are more in your favor.

Programs written in assembly language depend heavily on their comments to explain the program's inner workings. This has elevated the writing of good comments to something of an art form. (It's also quite time consuming.) In most cases, the comments show what the computer is doing, rather than what the program is doing.

Personal computing is about to leave the world of the horseless carriage and enter the world of the Model T.

I'd like to say a few words about some of the newer programming techniques that should be supported by a good programming language. These techniques include top-down design, modular design, and structured programming.

Top-down design is a method of solving a problem by starting at the highest, uppermost level, and working your way down. One way of telling if you are at the highest level is to see if you can express the entire problem in one sentence, a sentence such as, "Process monthly payroll."

Although most programs can't be designed entirely top-down, the process helps to bring an orderliness to software development. A very high percentage of assembly-language programs are written "bottom-up," starting with the subroutines, then working towards the higher levels.

In using modular design your program is broken down into modules, usually no longer than one page in length. Emphasis is on the "functional" nature of a module. Each module should perform a specific function or a closely related set of functions. One test to see if a module performs a function is to see if you can give it a name that begins with an imperative verb such as "test" or "compute."

The last of the new techniques is called structured programming. Briefly, it limits the number of basic structures that control the flow of a program (namely, the structures that control looping, branching, etc.) This restricted use of a programming language has been shown to produce programs that are more reliable and easier to maintain and modify. Structured programming also prohibits excessive use of GOTO statements since they make it very difficult to follow the flow of a program.

Top-down design is concerned with the chronology and hierarchy of program development, modular programming with the functional partitioning of the program, and structured programming restricts the permissible sequences of instructions to a few basic building blocks. Each of these is independent of the other. A good high-level language will allow the use of all three of these modern programming methods.

Now that we have decided that a good high-level language is the best thing since switching from octal to hexadecimal, let's take a look at some of the currently available offerings. Most of these fall into one of two categories:

> BASIC (Tiny through EXTENDED). This is an interpretive, interactive language. Originally designed for time-sharing use in schools; most versions closely adhere to the original Dartmouth BASIC, designed by Kemeny and Kurtz.
>
> PL/I derivatives (PL/M, MPL, PL/W, PL/Z, PL/M6800, etc.). These are basically subsets of IBM's PL/I, which runs on their larger machines.

A look at the hobbyist computer magazines reflects the current use of programming languages. Although they are supposedly simply for hobbyists, these magazines appear to be doing more to further the state of the art than the more traditional engineering journals.

Almost nothing is written in the PL/I-based languages. This is mostly because high cost and large equipment requirements have made them unavailable to any but the larger companies. (PL/M, for instance, requires an Intel development system, 64K of memory, and a dual-disk.) In addition, PL/I languages have been accused of producing as much as three times the amount of code as an equivalent assembly-language program. There is also some question as to whether a language originally designed to run on large computer systems is the best language for writing real-time computer application software.

What do we find written in BASIC? Games, number-crunching programs, some business software, and educational programs for computer-aided instruction. Most high school students these days have been exposed to BASIC in one form or another.

Enough of this canned software! It's time to flex our creative muscles.

Why no applications programs in BASIC? One reason is that BASIC is an interpretive language and its interpreter (requiring from 2K to over 8K bytes) must always be resident in the machine while the program is running. This would be quite a waste of memory in a controller application where the entire program may be less than 1K bytes of code. However, if the price of computer memory continues to fall, this may cease to be a serious objection. A bigger problem is the execution time of programs written in BASIC. Because of the internal workings of the BASIC interpreter, a program written in BASIC may run up to one hundred times slower than an equivalent assembly-language program. This places a serious restriction on BASIC for writing real-time programs.

BASIC has a few other problems. But it has almost none of the structures required for structured programming. A large program in BASIC will usually appear as a hopelessly complex mess to anyone but its author. Because of this unstructured nature, large BASIC programs can be exceedingly difficult to modify.

Moreover, BASIC uses only one or two characters to identify a variable, and it uses line numbers to represent locations within the program. This complete lack of self-documentation means that a well-documented BASIC program would probably need the same high percentage of comments as an assembly-language program.

TLC is a high-level language designed to fill the gap in the spectrum of currently available programming languages. TLC is not meant to be an "all-purpose" language like PL/I, nor is it meant to replace BASIC. It is aimed specifically at the area now occupied by assembly language, namely real-time systems and applications programs.

TLC is a simple language, easy to learn and use. It has control structures and data types that make it far more powerful than assembly language. Even though TLC is a high-level language, it will typically produce only about twenty percent more code than an equivalent assembly-language program.

One of the goals in designing TLC was that a TLC program should read like a "job description" for a computer—written in English. A program written in TLC is meant to be read by people. This is an important part of the TLC concept of what a high-level language should look like. A great deal of effort went into the human-engineering aspect of TLC. TLC is also designed to support the concepts of top-down design, structured programming, and modular programming.

Work on TLC began early in 1976, when I noticed that I was spending an alarming amount of time writing comments and otherwise trying to provide adequate documentation for large assembly-language programs. TLC evolved through actual use in writing applications programs. New structures and data types were created as the need arose.

Let's take a look at some of the details of TLC. First a word about names (the term *name* will be used interchangeably with *label*). In TLC, a name may be given to a variable, a constant, or a group of statements that form a procedure.

In TLC, a name can be as long as you want. In addition, colons can be embedded in the name to break it up into words. For example, you can have a variable named either *GROSSPAY*

ROMtutorial ROMtutorial

K: From kilo, which is in turn from the Greek *khilioi,* meaning a thousand. In computerese, however, K usually means 1024 (2^{10}) instead of 1000. Quantities expressed in binary, like memory addresses, are commonly figured in batches of 1024. For instance, 4K is equal to 4096.

Dual (floppy) disks: A floppy disk (also known as a diskette) is a flexible sheet of plastic with a magnetic coating, used for moderately high speed storage of data or programs for a computer. Although it looks like a 45-rpm record, it acts more like a magnetic tape. A dual floppy is simply two floppy disk drives in one cabinet.

Real time: Descriptive of on-line computer processing systems which receive and process data quickly enough to produce output to control, direct, or affect the outcome of an ongoing activity or process.

Byte: A piece of information consisting of eight bits. It has 256 possible values.

or *GROSS:PAY*. Using the colons is your option. The computer couldn't care less which name you use, but a human would probably prefer the second. Remember that if you put colons into a name, the result is still considered a single name.

One of the most important rules of programming style is the use of meaningful labels. Thus, a variable that will represent the day of the week can be called *DAY:OF:THE:WEEK* in TLC, instead of something like *WKDAY* in assembly language or just *W* in BASIC. This is an invaluable aid in documentation and greatly lessens the need for comments. A TLC statement such as:

```
SET NET:PAY=GROSS:PAY
              -TOTAL:
              DEDUCTIONS
```

can be considered self-documenting.

But here are some rules for making labels. The first character of the label must be a letter. Each remaining character must be either a letter, a number, or a colon. In TLC, command words are reserved and cannot be used as labels. We cannot have a label called *IF* since *IF* is a TLC command word. However, a reserved word can be embedded into a label:

```
TEST:IF:LAST:PAYCHECK
```

is okay since the *IF* does not appear by itself.

The basic unit of a TLC program is the TLC statement. This is a "command" that will be executed when the program is run. Only one TLC statement is allowed per line (multiple statements per line, as used in some other languages, are not allowed in TLC). However, a TLC statement can be longer than one line. This is done by putting a comma at the end of each unfinished line.

The *assignment* statement is used to change the value of a variable. All assignment statements begin with the command word *SET*. For example:

```
SET ALPHA=BAKER
SET ALPHA=3125
SET ALPHA="NOW IS THE
             TIME"
```

All of these examples set a variable to a new value. The previous value of the variable is lost. If we just want to alter the value of a variable, most high-level languages would look like this:

```
SET ALPHA=ALPHA + 1
```

This simply adds 1 to the value of *ALPHA*. This is valid in TLC, but an optional, shorter form can be used:

```
SET ALPHA + 1
```

which does the same thing.

An important concept in TLC is the *function-call.* At any point in the program, if there is a specific job to be done, and we have not yet figured out how, we write a label that describes this function. This is a function-call.

For example, in the middle of a payroll program, let's say the next thing for the program to do is to process the next paycheck. In TLC we write:

```
PROCESS:NEXT:PAYCHECK
```

What we are saying is "at this point, do this action." Notice that we have not specified any of the details of how this is to be accomplished. This allows us to proceed in a top-down fashion by saying what has to be done, rather than how. As a result, we are free to move on to the next part of the program without getting lost in details which are not important at this time.

The function-call is a great aid in the top-down development of a program. It allows you to define completely each level of the program before proceeding to the next lower level.

The function-call is used instead of the usual methods for calling subroutines, procedures, and macros in other languages. It is written according to our rules for making labels and must appear on a line by itself. It does not begin with a command word such as *DO, CALL,* or *GOSUB* because the function-call itself is considered to be a command. It also improves the readability of the program.

As we develop the lower levels of the program, each function-call has to be defined. It will either call a subroutine, a macro (a routine written in assembly language), or be replaced by one or two TLC statements that perform the function.

If there are any arguments that have to be passed to a subroutine, they are placed in a set of brackets following the function-call.

The program must be able to make decisions. This is done with a *conditional* statement in TLC. A conditional statement contains the command word *IF,* followed by a condition that will be tested at this point in the program. For example:

```
IF ALPHA=BAKER
```

If the test is *true,* all the statements that follow will be executed. The end of this conditionally executed block of statements is marked by the command word *FI* (*IF* reversed) on a line by itself.

```
IF THIS:STATEMENT=TRUE
   DO:THIS
   DO:THIS:TOO
FI
```

If the condition is *not true,* the program will skip right on to the *FI* and resume execution on the line that follows.

Note that the conditionally executed statements have been indented to make them stand out as a block. The *FI* is aligned directly under its matching *IF,* giving it the effect of a parentheses. The conditionally executed block of code can contain any number of statements.

We can also have a block of statements that are executed if the test is *false.* We use the command *ELSE* to denote this block of execute-if-false statements.

```
IF THIS:STATEMENT=TRUE
   DO:THIS
   DO:THIS:TOO
ELSE
   DO:THIS:OTHER:THING
FI
```

Is a language originally designed to run on large computer systems best for writing real-time software?

Note that the *ELSE* appears on a line by itself, and is aligned with its matching *IF* and *FI*.

TLC allows you to group a number of conditional statements together to test for more than one condition before executing a block of code. If *more than one* condition has to be *true,* we put the command *AND* in front of the added conditions:

```
IF THIS: STATEMENT = TRUE
AND IF THIS: STATEMENT
                    = TRUE
AND IF THIS: STATEMENT
                    = TRUE
     DO: THIS
FI
```

If you want the statements to be executed if *one of several* conditions is true, the conditional statements are joined together by the command *OR* between the statements on a line by itself:

```
IF THIS: STATEMENT = TRUE
OR
IF THIS: STATEMENT = TRUE
   DO: THIS
FI
```

You can use almost any combination of these conditional building blocks to make larger structures. This includes the ability to nest conditional statements to test for more complex conditions.

Another structure in TLC is the *loop.* It repeats a sequence of instructions endlessly until told to quit. An example of a loop in TLC is:

```
LOOP
      DO: THIS
      DO: THIS: TOO
POOL
```

The *LOOP* marks the beginning of the structure. The block of statements to be executed by the loop is indented, as with conditional statements. The *POOL* marks the end of the loop. As with *IF* and *FI, LOOP* and *POOL* act like parentheses. When *POOL* is reached, the program returns to *LOOP* and begins again.

There are two ways to terminate the execution of a loop. The first is by using a conditional statement beginning with the command word *EXIT:*

```
LOOP
      DO: THIS
      EXIT IF THIS:
            STATEMENT = TRUE
      DO: THIS
POOL
```

If the condition in the *EXIT* statement is *true,* execution of the loop is terminated, and the program continues at the first statement *after* the *POOL.*

The second method for terminating the loop is by giving it a parameter enclosed in brackets, to indicate how many times the loop should be executed:

TLC was designed as a real-world programming language, so it has facilities for interacting with different situations.

```
LOOP [15]
      DO: THIS: THING
POOL
```

In the example above, the loop will be executed 15 times and the program will continue on the line following *POOL.* This type of loop is valuable if we know in advance how many times the loop should run. The loop parameter can be a variable, a constant, or an arithmetic expression.

Loops can have a parameter and an *EXIT* statement(s). In this case, the loop will terminate as soon as either condition is met (as soon as either the parameter is finished or the *EXIT* statement is *true*).

Loops, like conditional statements, can be nested to form more complex structures. An *EXIT* statement in a nested loop, when true, will leave its current loop and exit to the next higher level.

In modular programming, the program is broken down into smaller, functional modules called *procedures.* This is done even though the function performed by the procedure is used only once in the program.

On the other hand, *subroutines* are used when the same block of statements is needed several times in a program. Only one copy of the subroutine is in the program and it is "called" whenever it is needed. A subroutine, for instance, could compute the cosine of a number. In TLC, procedures and subroutines are treated in the same way and the terms will be used interchangeably.

The first line of a TLC procedure begins with a dollar sign ($) followed by the name of the procedure. This is the entry point of the procedure. Execution of the procedure begins at the first statement after the label.

The body of the procedure can consist of any number of TLC statements, but a good rule is to limit the length to less than a page, so as to preserve the modularity of the program. The end of the procedure is marked by the command word *RTN,* on a line by itself. Here's a sample procedure:

```
$ PROCESS: NEXT: PAYCHECK

COMPUTE: GROSS: PAY
COMPUTE: TOTAL:
                 DEDUCTIONS
COMPUTE: NET: PAY
PRINT: PAYCHECK
UPDATE: RECORDS

RTN
```

When the program encounters the function-call:

```
PROCESS: NEXT: PAYCHECK
```

it will transfer control to the procedure that we have defined to perform this function. In the example above, the procedure is composed of a group of function calls. Each of these will be defined later. This feature of TLC is designed to aid in the top-down development of programs.

When the program reaches the *RTN* at the end of the procedure, the program resumes execution at the first statement *after* the function-call.

There are some functions, for instance initializing the stack pointer for a particular microprocessor, that are completely machine-dependent and have no equivalent in TLC. In this discussion, we define a *macro* as a section of the program that is written directly in the microprocessor's assembly language. A macro is written in the same fashion as a subroutine, except that an exclamation point is used in front of its label:

```
! INITIALIZE: STACK: POINTER

LDS #H7FFF
TSX
STX SPSAVE

RTN
```

Again, note that while a procedure is written in TLC, the macro is written in assembly language. As with a procedure, the macro is called by a function-call (putting its name on a line by itself). Thus, when reading a TLC program, there is no way of knowing if a particular function-call is implemented as a subroutine or a macro. The reason for this is to mask the exact nature of the lower level of the program from the higher level. The purpose of the function-call is to express *what* the program will do at certain points, rather than *how.*

When the TLC program is translated into assembly language, all of the assembly-language instructions that are listed in the macro are copied into the assembly-language version at every point that the name of the macro appears in the TLC program.

There is another difference between a procedure and a macro. There will be only one copy of a procedure in the assembly-language version and it can be called from a number of different points in the program. On the other hand, if a particular macro is called ten times in a TLC program, there will actually be ten copies of it in the assembly-language version. The macro facility also allows us to optimize the running time of the more important parts of the program without sacrificing the documentation.

In TLC, any variable or arithmetic expression that appears in parentheses is a *pointer.* Instead of being used as data, it points to where the data can be found. This lets us access tables, arrays, matrices, etc. For instance:

```
SET ALPHA = (ADDR TABLE:
                   START + 25)
```

In the example above, the command word *ADDR* is used to refer to the address of a variable, rather than its contents. The statement above sets *ALPHA* to the value of the 26th entry of a table beginning in memory at the address *TABLE: START.* Another feature of TLC pointers is that they may be nested.

All we need now are some commands to transfer data in and out of the computer. Let us assume that the computer is attached to a Teletype. The *INPUT* command lets the user enter data from the Teletype keyboard. The *PRINT* command causes data to be printed on the Teletype.

This data can be in several forms, so all input and output statements must show the data type. These are a few of the TLC data types:

'*DNUM*' = decimal number
'*HNUM*' = hexadecimal number
'*CHAR*' = ASCII character

Character strings are denoted by enclosing them in quotes. Here are a few samples:

```
PRINT DNUM = GROSS: PAY
PRINT HNUM = STACK:
              POINTER: ADDRESS
PRINT "NOW IS THE TIME"
INPUT CHAR = COMMAND:
                          CHAR
```

The last example waits for you to type a key on the Teletype and stores its value in a variable called *COMMAND: CHAR.*

Programming can be considered a two-part process. First, we must design the task for the computer, specified to the lowest possible level, while using a machine-independent language. Once the program has been designed, we translate the high-level program into the machine language of a particular computer. This "partitioning" greatly simplifies the programming process.

Unfortunately, the assembly-language programmer has to do his programming in a "bottom-up" process. The average assembly-language programmer creates his program in tiny fragments, expressing them directly in the instruction set of a particular computer, then tacks on some comments that he hopes will give a clue as to how the program works.

With TLC, on the other hand, the two-part method is used. First, the TLC version of the program is refined to the point where it has been specified to the lowest practical level. With TLC as the language and the use of the programming techniques discussed earlier, this part will be a lot less painful than working directly in assembly language.

The second step is to translate the TLC program into a *target* language. In most cases, the target language will be the assembly language of a particular computer. However, there is no reason why TLC programs cannot be translated into BASIC or other high-level languages. Adding TLC to BASIC would aid in writing modular, structured programs in BASIC. In any case, the TLC program is translated *mechanically* into the target language according to a set of predefined rules.

As an example, look at a sample program, *TEST THE MEMORY,* that tests a designated part of a computer's memory. The test is designed to spot faults including bad memory chips, shorted data lines, and shorted address lines.

It does the test in two *phases.* First, it fills the test area of the memory with a special pattern of data. In the second phase, it reads back the data and counts the number of faulty memory locations.

As we will see, the program is broken down into modules (procedures) that also represent lower levels of the program. The first module, *$ TEST: THE: MEMORY,* is the highest level. The program begins its execution on the first line after the label. The first program statement, Line 2, is an assignment statement that initializes a variable.

The next statement is the beginning of a loop. The block of statements that will be executed each time through the loop is denoted by indenting the entire

TLC can teach the fundamentals of programming; it can also be a medium for exchanging programs and ideas.

group. Inside the loop (Line 4) we initialize another variable, and on Line 5 the function-call *DO: ONE: TEST* calls the next lower level of the program. This call has been implemented as a procedure, listed right after the first module. When the function-call is reached, control passes to the procedure with the same label.

Let us say, for the moment, that we have not decided exactly what the test will consist of. However, writing the function-call *DO: ONE: TEST* will remind us later exactly what action has to be performed and where in the program it should occur.

After the procedure *$ DO: ONE: TEST* has finished its job, it will return control to the next line (6) after the function-call. The group of conditional statements (Lines 6-10) will then print *"OK"* if no errors were found, or else will print the number of errors (as a decimal number). The program then increments the starting value for the next test (Line 11).

The *POOL* (Line 12) marks the end of the loop and returns to the *LOOP* above it. Note that this loop has no parameter or *EXIT* statement. This is an endless loop that, in theory, would run forever.

Now we have pretty much defined the upper level—initialize, do a test, print the results, do another test, etc. Note that the very highest level of the program *TEST: THE: MEMORY* is also the name of the first module of the program.

Now we can define *$ DO: ONE: TEST.* Note that it also contains a loop, and has the function-call *DO: ONE: PHASE,* which is the next lower level of the program. This loop has an *EXIT* statement that terminates the loop after the second pass.

The next lower level, *$ DO: ONE PHASE* follows the same pattern of the modules before it. The loop calls *$ DO: ONE: BLOCK,* which is the next lowest level. The *EXIT* statement will terminate the loop when the last memory location is reached.

The procedure *$ DO: ONE: BLOCK* is the lowest level of the program. It does the real dirty work, so watch closely. First, note that the loop has both a parameter and an *EXIT* statement. The effect is "do this until the condition in the *EXIT* statement is true, but not more than 256 times." Remember that the memory test performed by this program is done in two phases. Up to this point, it has not mattered which phase was being executed. At this point (Line 39), we encounter a conditional statement that says "if this is phase one, load the memory, else (must be phase two) test the memory." Note that the memory location being tested is accessed by using a pointer, which just happens to be named *POINTER.* On Line 40, the pointer is enclosed in parentheses, which means that it is pointing to a memory location. On Line 48, we move the pointer to access the next location by adding one to its value. In this case, we did not use the parentheses, so we were talking about the pointer itself.

A "prettyprint" feature was considered so important that it was first to be implemented.

Line 42 is the beginning of another conditional statement. This statement is *nested* into another conditional statement. The " \ " in front of the equal sign means *NOT* and the combination is read *NOT EQUAL.* During the second (test) phase, this piece of the program counts the number of errors found.

Now let us trace the flow of the program backwards, to see how it ends. In each phase, when *$ DO: ONE: BLOCK* reaches the point where it has processed the last memory location, it returns to *$ DO: ONE: PHASE,* which returns to *$ DO: ONE: TEST.* If this was phase one, the test is finished and we return to *$ TEST: THE: MEMORY.* Now we print the results and get ready for the next test. And so it goes.

To give the next illustration, we will introduce a TLC data type called a *Bit Pattern.* Imagine that on a piece of equipment being controlled by a computer there is a group of eight toggle switches. These switches are interfaced to the computer through some sort of parallel input port. To the computer, they look like any other group of eight-bit binary data.

Now suppose we are interested in one particular on-off pattern of these switches. Suppose further, that we are only interested in four of the switches. At this time we do not care about the position of the others. It would be nice if we could give this combination a name, as an aid to documentation. We can do this by treating the data represented by the switches as a Bit Pattern.

The Bit Pattern looks like a binary number, but instead of ones and zeros it is made up of these three characters:

'T' means TRUE
'F' means FALSE
'0' means DON'T CARE

A Bit Pattern is denoted by a sharp sign (#) in front of it. Now we can give a name to the particular condition that is represented by this Bit Pattern:

```
EMERGENCY: VALVES: OPEN
          = #TF00: 00FF
```

Whenever the name *EMERGENCY: VALVES: OPEN* is used in the program, only the positions marked by 'T' and 'F' will be used.

Now we give a name to the location from which the switch status is fetched. We will call this location *CONTROL: PANEL.*

All of this may seem a little devious, but consider this piece of program:

```
IF CONTROL: PANEL
  = EMERGENCY: VALVES: OPEN
    SET ALARM: READINESS
                    = TRUE
ELSE
    SET ALARM: READINESS
                    = FALSE
    BLINK: RED: PANEL: LIGHT
    ACTIVATE: WARNING:
                    BUZZER
FI
```

The idea of the Bit Pattern came up while designing complicated interfaces for a driving simulator and gave me the ability to name things like gearshift positions or a burned clutch where a combination of switches represented a particular condition. I have included some of the Bit Patterns used in one of the driving simulator programs, as well as a portion of a program that uses these Bit Patterns. Note the use of colons (as in labels) to break one-byte binary numbers into easy-to-read nibbles.

Since TLC was designed as a real-world programming language, it has a number of facilities to help it interact with different situations. The *WAIT* command can be used to suspend program operation for a precise amount of time (six-microsecond resolution). This command was used to generate a series of pulses for a four-channel radio control system in a driver-training simulator. The delay time is enclosed in brackets (as with other parameters). If no parameter is given, the program will wait indefinitely.

TLC also has commands to implement a multi-tasking or multi-user facility. The *INIT* command sets up a new job, gives it a stack for subroutine returns and its own data area to avoid conflicts with other users. The *FREEZE* command suspends the operation of a task as might happen if it's waiting for you to type a key on your terminal, or some other I/O operation. When the *FREE* command is executed, the task is restored at the exact point where it stopped.

The *FREE* and *FREEZE* command were used in a computer-based security system that watches three buildings in New York City. The system interacts with a variety of peripherals including keyboards, CRT displays, modems, and a real-time clock. The TLC multitasking facility allows the entire system to run without any interrupts.

Most high-level languages are translated to a lower level by a compiler, an interpreter, or a macroprocessor.

In the case of TLC, none of these has been fully implemented. All that exists at this point is a "smart" text editor program that aids in writing TLC programs and will eventually grow into a compiler. One of the features of the text editor is that it will automatically format TLC programs by printing them with the proper indentation as shown in the examples. This "prettyprint" feature was considered so important that it was the first to be implemented. The indentation is a great aid in showing the exact structure of the code.

At this time, TLC programs are hand-translated into the target language by using a set of pre-defined rules. This may sound primitive, but it is not so bad when you consider the alternatives.

First of all, the translation is *mechanical,* and therefore not very time-consuming. Secondly, since the program is translated according to a set of rules, there is no need for comments on the assembly language version. If the computer only was to assemble programs that are already completely stored in its memory, dropping the comments can double the size of the programs that it can handle.

During translation, each TLC statement is treated as a separate entity. It is translated by itself, without regard for what a previous statement may have left in the computer's registers. This means that you may end up loading a register with data that was already there. This slight loss of memory utilization yields a far bigger benefit in terms of reliability. If a translated statement is modified at a later date, there will be fewer side effects on the statements that follow. After the assembly-language version has been completely tested and debugged, redundant instructions can be removed if memory utilization is an absolute must.

TLC was designed for computer programming, but it has other uses besides this primary function. With its inherent simplicity and clean structure, TLC can be used as a language to teach the fundamentals of programming. It can also be used as a medium for exchanging programs and ideas.

As I mentioned earlier, TLC evolved through actual use. Programs that have been written in TLC include a relocating loader, a monitor that operates along with a real-time clock, as well as the simulator and security systems. The TLC compiler itself is being written in TLC.

TLC has most of the advantages claimed for high-level languages. Programming time has been shortened, as has the time required for testing and debugging. The programs are easier to modify and a high percentage of them run the first time.

Inspired by our current economic policies, I would like to propose the concept of "zero-base" programming. There must be a continuous re-examination of all parts of a computer system to see how they affect programming. The first question for tomorrow is: *IS TLC OBSOLETE?* ▼

ROMtutorial ROMtutorial

Parallel input port: A port is some arrangement for getting data signals into or out of a computer. An input port gets them in. A parallel port is one that has the signals running in parallel—meaning that a number of signals travel through the port at once. (The alternative is a serial port, through which the signals travel one after the other.)

Bit: The simplest unit of information in computing—the condition of being either on or off. Electrical circuits are well-suited to dealing in bits, since it's no problem (usually) to tell the difference between a high and a low voltage.

Binary system: A system of numbers based on the two digits 1 and 0 instead of using strings of digits which can have ten different values (0,1,2,3,4,5,6,7,8,9) as in the decimal system. The columns are no longer ones, tens, hundreds, etc., but are instead ones, twos, fours, eights, etc., and are called bits. Binary is very useful where machines have to handle numbers. It is also known as the base-two system.

Interface: The boundary between different sections of a computer, or between the computer and the outside world. The circuitry crossing the boundary is also called the interface.

Keyboard: A device like a typewriter keyboard that enables a person to key information into a computer. The information, in sequences of characters (letters, digits, and special symbols) is called text strings, or simply text.

CRT: Cathode ray tube. An electronic vacuum tube containing a phosphor-coated screen on which information may be displayed by the use of controlled beams of electrons; the electrons strike the screen coating, causing it to emit light. Television sets use them.

Modems: Devices which convert computer data signals into audio signals which can be sent over telephones.

Interrupt: Relates to the suspension of normal operations or programming routines of microprocessors. Most often designed to handle sudden requests for service or change.

TLC Vocabulary

```
**** TLC VOCABULARY ****   (C) 1977 TONY KARP



** CONDITIONAL

IF
ELSe
FI
AND
OR



** LOOPS

LOOP                    *START OF LOOP
LOOP [EXPRESSION]       *EXPRESSION IS NUMBER
                        *OF ITERATIONS
                        *NO INDEX IS CARRIED
EXIT IF                 *CONDITIONAL EXIT FROM LOOP
EXIT                    *UNCONDITIONAL EXIT FROM LOOP
POOL                    *END OF LOOP



** DATA TYPES

* SIZE OF WORD CAN BE SPECIFIED
* DEFAULT IS WORD SIZE OF TARGET MACHINE
* NUMBER CAN BE DECLARED SIGNED, UNSIGNED
* DEFAULT IS UNSIGNED

HNUM        *HEXADECIMAL NUMBER
DNUM        *DECIMAL NUMBER
BNUM        *BINARY NUMBER
SIGNED      *NUMBER WILL BE TREATED AS
            *TWO'S COMPLEMENT
BCD         *2 BCD DIGITS PER BYTE
TEXT        *ASCII TEXT STRING
INT         *MOST SIGNIFICANT HALF OF NUMBER,
            *REGARDLESS OF SIZE
FRAC        *LEAST SIGNIFICANT HALF OF NUMBER,
            *REGARDLESS OF SIZE
CHAR        *ASCII CHAR
ADDR        *ABSOLUTE ADDR OF ANOTHER
            *LABEL

** ARITHMETIC OPERATORS

+
-
M   *MULTIPLY
D   *DIVIDE



** RELATIONAL OPERATORS

=
>
<
\   *NOT



** LOGICAL OPERATORS

TO:INCLUDE     *INCLUSIVE OR
CLEARED:BY     *LOGICAL AND
REVERSED:BY    *EXCLUSIVE OR
REVERSED       *COMPLEMENT



** ASSIGNMENT

SET     *INITIALIZE OR CHANGE THE VALUE OF A
        *VARIABLE
SWAP    *EXCHANGE THE VALUES OF TWO VARIABLES



** SUBROUTINE

$ SUBROUTINE:NAME    *SUBROUTINE ENTRY
SUBROUTINE:NAME      *SUBROUTINE CALL
RTN                  *RETURN FROM SUBROUTINE
RTN [INTERRUPT]      *RETURN FROM INTERRUPT



** MACRO CALL

! MACRO:NAME         *MACRO DEFINITION
MACRO:NAME           *MACRO CALL
RTN                  *END OF MACRO

** I/O AND FORMAT

* 'INPUT' AND 'PRINT' MUST BE FOLLOWED
* BY DATA TYPE.

INPUT
PRINT

SKIP                 *LINE FEED
SKIP [EXPRESSION]
SPACE
SPACE [EXPRESSION]



** STACK

PUSH [POINTER:NAME]
PULL [POINTER:NAME]



** PROGRAM CONTROL

WAIT *WAIT FOREVER
WAIT [HOW:LONG]
INIT [STACK:NAME]
FREE [STACK:NAME]
FREEZE



** MISC

$     *START OF SUBROUTINE
!     *START OF MACRO DEFINITION
```

```
,       *CONTINUE THIS STATEMENT ON NEXT LINE
(       *POINTER
)
[       *PARAMETER
]
:       *HEX NUMBER
#       *BINARY NUMBER
*       *COMMENT
“       *TO ENCLOSE TEXT STRING
.
** COMPILER DIRECTIVES

END     *END OF PROGRAM
ABS     *ABSOLUTE ORIGIN
REL     *RELOCATABLE ORIGIN
IFT     *START OF CONDITIONAL SECTION
1FF     *START OF CONDITIONAL SECTION (FALSE)
ENDC    *END OF CONDITIONAL SECTION
RES     *RESERVE MEMORY
DATA    *PLACE DATA AT THIS LOCATION
DCL     *START OF DECLARATIONS
LCD     *END OF DECLARATIONS


                END
```

Test the Memory

```
$ TEST:THE:MEMORY
  SET TEST:WORD:START:VALUE = 0
  LOOP
      SET ERROR:COUNT = 0
      DO:ONE:TEST
      IF ERROR:COUNT = 0
        PRINT "OK"
      ELSE
        PRINT DNUM = ERROR:COUNT
      FI
      SET TEST:WORD:START:VALUE + 1
  POOL
  RTN


$ DO:ONE:TEST
  SET TEST:PHASE = 0
  LOOP
      SET TEST:PHASE + 1
      DO:ONE:PHASE
      EXIT IF TEST:PHASE = 2
  POOL
  RTN


$DO:ONE:PHASE
  SET POINTER = STARTING:LOCATION
  SET TEST:WORD = TEST:WORD:START:VALUE
  LOOP
      DO:ONE:BLOCK
      EXIT IF POINTER = END:LOCATION
      SET TEST:WORD + 1
  POOL
  RTN


$ DO:ONE:BLOCK
  LOOP [256]
      IF TEST:PHASE = 1
        SET (POINTER) = TEST:WORD *LOAD MEMORY LOCATION
      ELSE
        IF (POINTER) \ = TEST:WORD *TEST MEMORY LOCATION
          SET ERROR:COUNT + 1
        FI
      FI
      EXIT IF POINTER = END:LOCATION
      SET TEST:WORD + 1
      SET POINTER + 1
  POOL
  RTN
.
```

Bit Patterns

```
** BIT PATTERNS



*** PIA #2, A SIDE

COLLISION          #0000:000T
CLUTCH:POP         #0000:00T0
CLUTCH:BURN        #0000:0T00
STALL              #0000:T000
NO:ALARM:FAULT     #0000:FFFF




** PIA #2, B SIDE

TURNED             #0000:000F
PRESSED            #0000:00T0
DOWN               #0000:0F00
FIRST              #0000:T000
SECOND             #000T:0000
REVERSE            #00T0:0000
NEUTRAL            #00FF:F000
BRAKE:SET          #0T00:0000
ENGAGED            #T000:0000

```

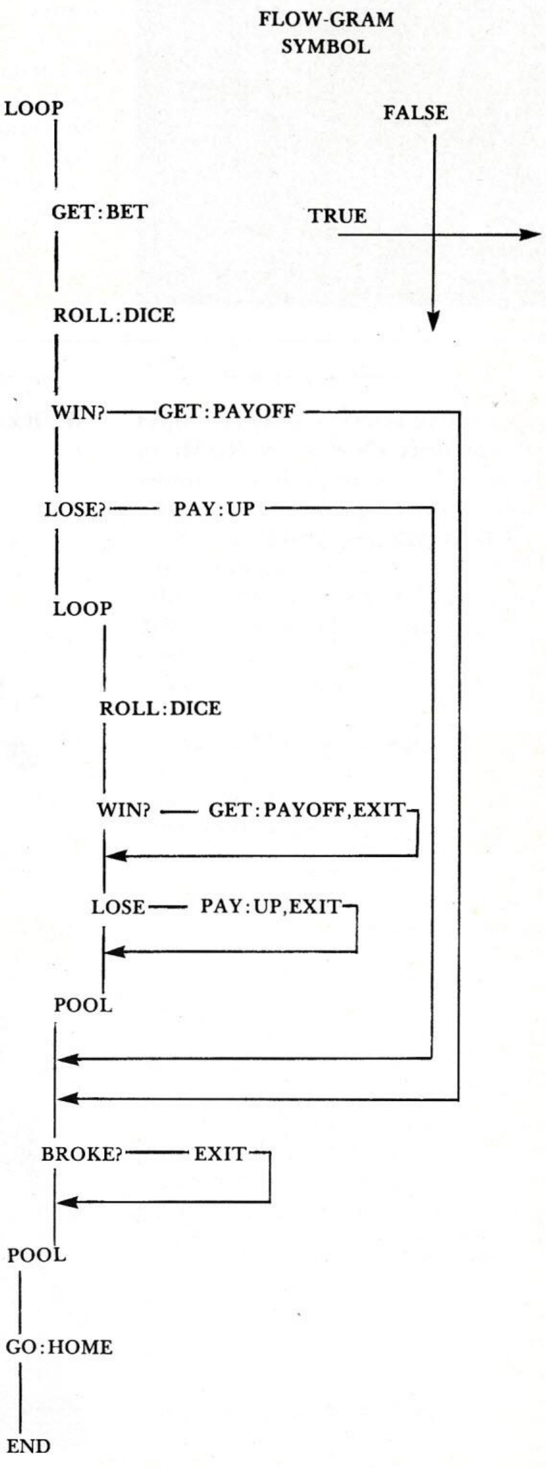
Evaluate Simulator Status

```
$ EVALUATE:SIMULATOR:STATUS

  IF ALARM = COLLISION
    CLEAR:AND:INIT:CRT
    SET CRT:LINE:1 = "HEY MAN, YOUR TRUCK IS STUCK!!"
    SHUT:DOWN:SYSTEM

  ELSE
    IF CLUTCH:PEDAL \= RIDING
      RESET:CLUTCH:RIDE:TIMER

    ELSE
      IF CLUTCH:RIDE:TIMER = FINISHED
        CLEAR:AND:INIT:CRT
        SET CRT:LINE:1 = "GET YOUR FOOT OFF THE CLUTCH!!"
        SET CRT:LINE:3 = "IT'S NOT A FOOT REST!!"
        SHUT:DOWN:SYSTEM

      ELSE
        IF MOTOR:RUNNING = FALSE
        DO:MOTOR:START:ROUTINES

      ELSE
        IF IGNITION:KEY = TURNED

          IF REV:UP:TIMER = FINISHED
            DO:MOTOR:RUNNING:ROUTINES
          FI

        ELSE
          SET MOTOR:RUNNING = FALSE *KEY TURNED OFF
          SET RESTART:MOTOR = TRUE
        FI
      FI
    FI
  FI
FI

RTN

.
```

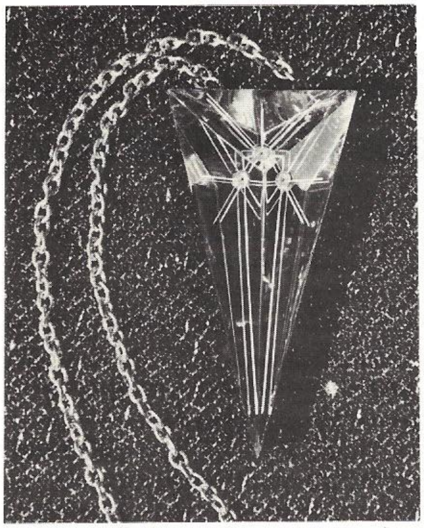
Craps in TLC

```
** CRAPS IN TLC      (c) 1977 TONY KARP


LOOP
    GET:BET
    ROLL:DICE
    SET DICE:TOTAL = DIE:1 + DIE:2

    IF DICE:TOTAL = 7
    OR
    IF DICE:TOTAL = 11
      SET BANKROLL + BET

    ELSE
      IF DICE:TOTAL = 2
      OR
      IF DICE:TOTAL = 3
      OR
      IF DICE:TOTAL = 12
        SET BANKROLL - BET

      ELSE
        SET WINNING:POINT = DICE:TOTAL

        LOOP
            ROLL:DICE
            SET DICE:TOTAL = DIE:1 + DIE:2

            IF DICE:TOTAL = WINNING:POINT
              SET BANKROLL + BET
              EXIT
            FI

            IF DICE:TOTAL = 7
            OR
            IF DICE:TOTAL = 11
              SET BANKROLL - BET
              EXIT
            FI
        POOL

      FI
    FI

   EXIT IF BANKROLL<CARFARE
POOL

GO:HOME

      END
```

Craps Flow-Gram


FLOW-GRAM
SYMBOL
FALSE
TRUE
LOOP
GET:BET
ROLL:DICE
WIN?
GET:PAYOFF
LOSE?
PAY:UP
LOOP
ROLL:DICE
WIN?
GET:PAYOFF,EXIT
LOSE
PAY:UP,EXIT
POOL
BROKE?
EXIT
POOL
GO:HOME
END

Run On Micros Run On Micros Run On



ALL SPARKLING AND BRIGHT

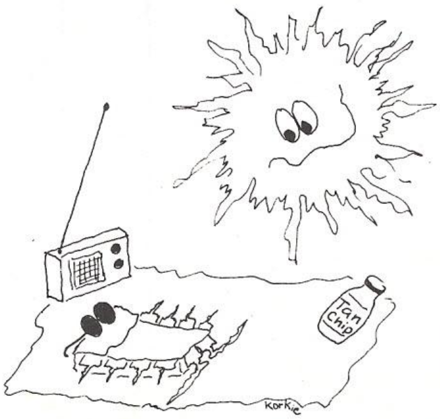
Remember Ted Nelson's column in last month's *ROM?* Well, that hardware jewelry looks as if it will be around a lot earlier than 1982. Consider the Star Jewel, for instance. A hearing-aid-battery-powered LED blinks through the multifaceted colored lucite "jewel" (red, green, amber, blue). "As the pendant swings lightly on its chain (it only weighs an ounce), the jewel seems to dance with inner reflections."

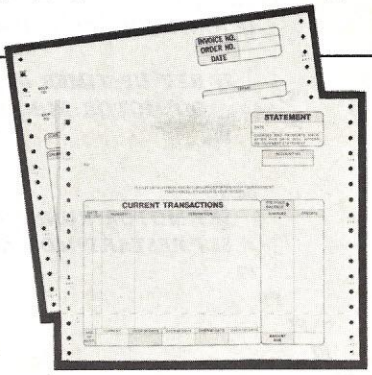
ATRA
P. O. Box 456
Minneapolis, MN 55440

BUREAUCRATIC SUNSHINE

If you're working away at Project Prometheus (November *ROM*) or some other solar project, consider some financing from Uncle Sam. HUD is offering grants of up to $250,000 for solar demonstration projects. These are aimed at solar energy use at the common-man level, and there could hardly be a more practical application for your micro.

National Solar Heating and
Cooling Information Center
P. O. Box 1607
Rockville, MD 20850





MAKING MONEY WITH BILLS

One of the problems with microcomputers for the small businessman is that the suppliers of such necessities as business forms, continuous number 10 envelopes, and pressure-sensitive labels aren't exactly overjoyed to service accounts ordering small quantities. In fact, usually they won't even bother to talk with a customer if he's talking in terms of just a couple of thousand imprinted forms. All that is changing, however, thanks to the Kenmor Co. They will supply you with a wide line of regular and window envelopes, invoice/statement forms, as well as custom sheets, plain or imprinted with your logo. Sprocketed, perforated, and ready to run on your printer, they may be ordered in quantities as small as one thousand.

Kenmor Co.
675 McLean Ave.
Yonkers, NY 10704

SPEAK EASY

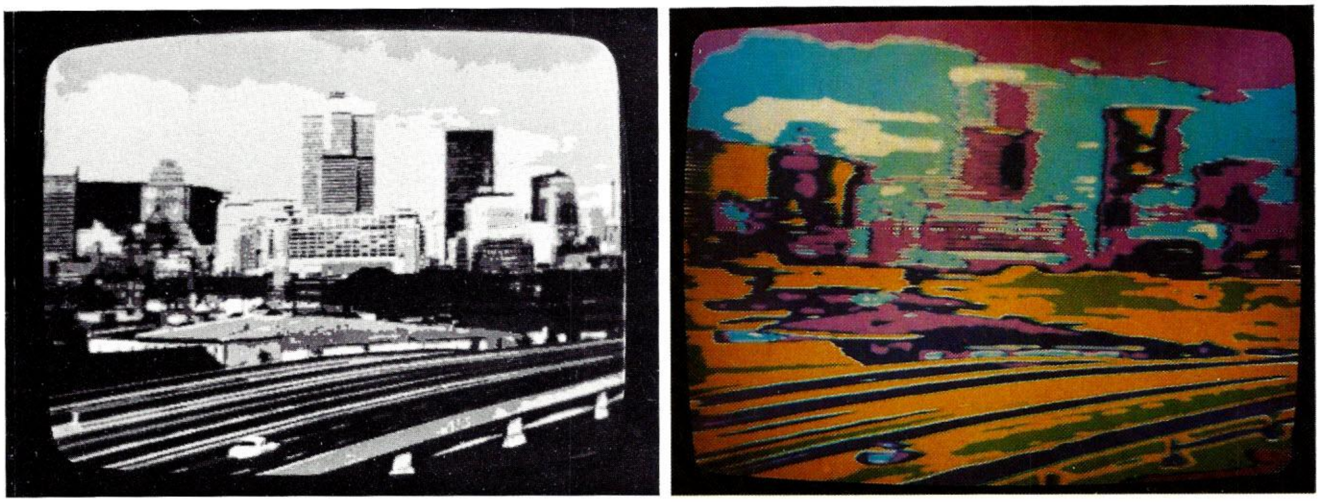
We all know the classic microcomputer question, "But what do you do with it?" Well, if the answer is sometimes still hard to find, there's no doubt that there *are* answers. But they may well appear tomorrow in a form quite different from what we expect today. Take voice synthesizers, for instance. Almost everyone wants one for his home computer. The novelty of a talking machine is immense. Yet from an actual do-something-with-it standpoint, voice synthesizers may well have their most important function as dedicated units.

Consider the Phonic Mirror or HandiVoice, being marketed by American Hospital Supply Corp. Here is the Votrax speech synthesizer in a portable, battery-operated mode. With a self-contained vocabulary of nearly five hundred words and phrases plus phoneme capabilities, a nonverbal/nonvocal human being suddenly gains a voice.

A touch-sensitive display board on the HC110 has 128 stations with four available overlays representing words, pictures, symbols, and controls. When the station is touched, Phonic Mirror HandiVoice speaks what is written or illustrated.

American Hospital Supply Corp.
One American Plaza
Evanston, IL 60201

GRAPHICS GALORE



When it comes to inexpensive graphics, the Matrox ALT-256**2 is hard to beat. The card contains all interface electronics, a TV-sync generator, and its own 65,536 x 1 refresh memory.

Compatible with any S-100 bus computer, the board outputs a high-resolution (256 x 256 dot raster) composite video signal. The complete screen can be cleared or preset with a single instruction.

The MTXGRAPH software subroutine package provides for initialization, screen erase, single-point display and erase, as well as end-point vector line generation and deletion—all in less than 1K.

Multiple ALT-256**2 cards can be combined to yield color/gray-scale pictures. This picture, for instance, was produced by feeding the outputs of three cards to the red, blue, and green inputs of a color monitor. The resulting pseudo-color pictures can have any of eight colors arbitrarily assigned to each gray level in a manner similar to the way NASA processes many of its pictures.

Matrox Electronic Systems
P. O. Box 56
Ahuntsic Stn.
Montreal, Que. H3L 3N5
Canada

0101
0100
0011
0010
0001

BLASTOFF

2005AD, Inc.
2005 Naudain Street
Philadelphia, PA 19146

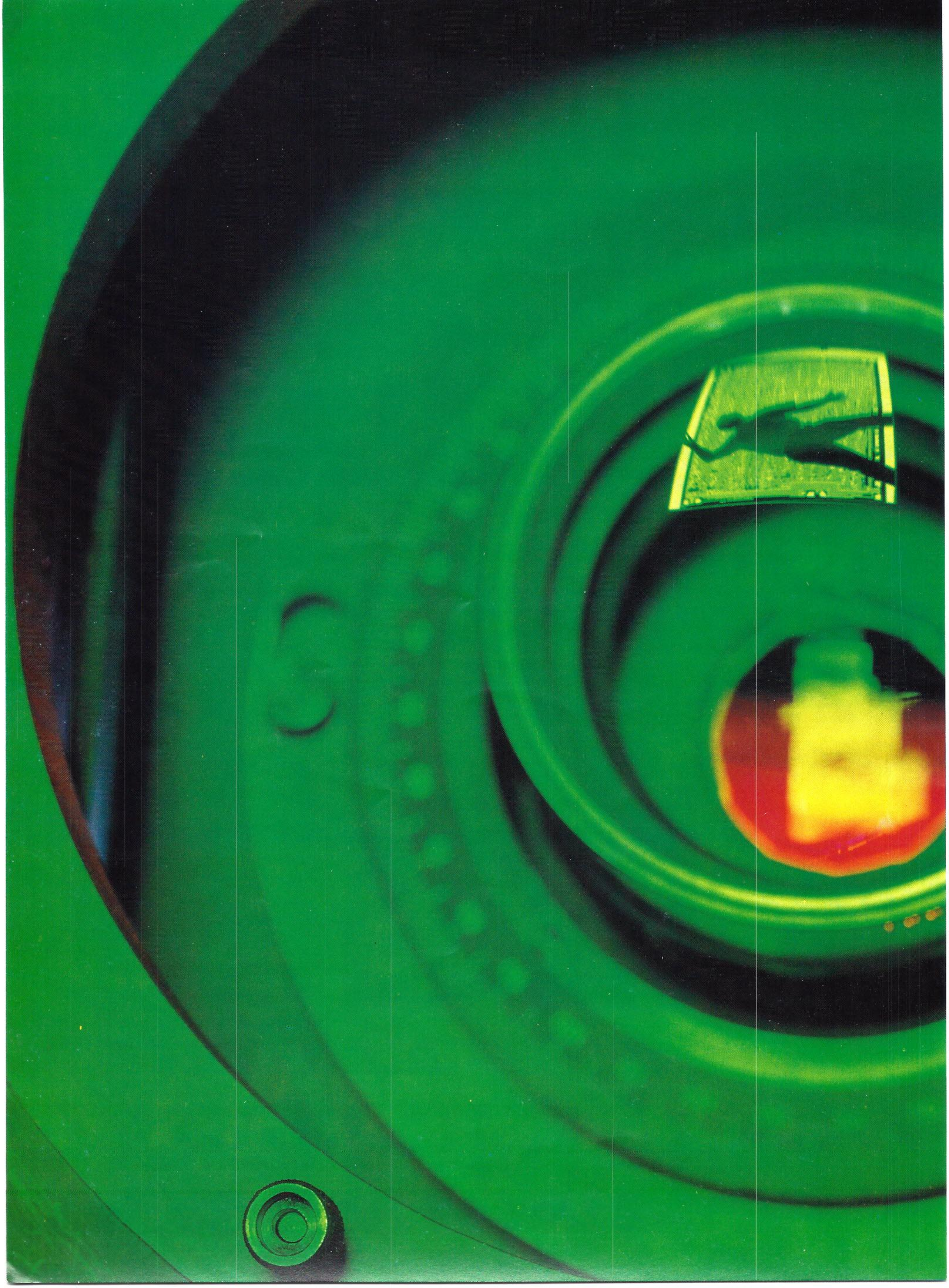
SFS WALLETSIZE

If microcomputers are the ultimate toy, then SFS Walletsize is surely the ultimate twentieth century fantasy. Here in a compact four-feet-square-by-five-feet-tall enclosure is a complete microprocessor-based space-shuttle simulator. Put it in your basement or garage—the kids' neighborhood clubhouse will never be the same again.

Slip into the pilot's seat, slide into the cabin, and you become Buck Rogers, Flash Gordon, and Luke Skywalker all rolled into one. Avionic instruments and three video monitors constantly monitor your space flight. But controlling the SFS Walletsize through your flights in fantasy space is up to you.

There's a lunar landing program, of course. Everyone needs a little localized practice to polish flight skills before entering hyperspace. And, although if you fail to touch down properly the screen displays the usual caustic comments about your qualifications, in Walletsize's 3-D mode Lunar Landing becomes a totally new game. Then there is the 3-D docking program involving the pilot in roll, pitch, and yaw on the x, y, and z coordinates.

Written in PolyMorphic BASIC (A00), the simulation can be reprogrammed for a wide variety of other space adventures. There's 40K of memory and more goodies than you can shake a Jawas at.



January 1978

ROM

COMPUTER APPLICATIONS FOR LIVING

This month's ROM centerfold shows an ultraminiaturization camera lens, reflecting one of the first steps in the manufacture of a microprocessor. Here a mylar circuit pattern measuring several feet square is photoreduced to a size about a quarter of an inch square. The camera is mounted on a suspension system, independent of the building in which it is housed, to minimize vibration and to obtain the best resolution possible.

Photograph courtesy of National Semiconductor

Seven points to consider before you buy your small computer.

In this magazine, alone, there are probably a dozen ads for small computers. New companies are breaking ground like spring flowers.

How, then, do you determine which computer offers the features you need most . . . at the price you can afford?

We'd like to propose seven basic questions to help you make an intelligent decision.

1. How complete is the computer system?

Many buyers of small computers are in for a rude awakening when they have to spend additional money for interfaces.

The Sol-20 Terminal Computer was the first *complete* small computer system. Everything you need to make it work is included in the basic package.

2. Is powerful system software available?

It won't do if your system is "tongue-tied."

Processor Technology Corporation has devoted more effort to the development of software than any other small computer maker. Our latest offering is the first fully implemented disk operating system for a small computer: PTDOS. It contains over 40 major commands, several languages and numerous utilities. Our high level languages include Extended BASIC, Assembler, FORTRAN,* FOCAL and PILOT.*

3. Is the system easy to expand?

More and more computer owners are expanding their small computers to handle business and other specialized requirements.

The largest Sol system can handle 64K bytes of RAM memory and operate with a three megabyte on-line disk memory. Sol systems use the S-100 Bus. So you can use a wide variety of hardware.

4. Is the computer well-engineered?

Our Sol systems are the most conservatively rated and ruggedly built in the industry, period. In addition we designed them with you, the user, in mind; Sols are easy to build and a joy to operate.

5. Does it have proven reliability?

What is the track record? There are over 5,000 Sol systems in the field. Our track record for reliable performance is unparalleled in the small computer field.

6. Does it have good factory support?

A computer is a complex piece of hardware. So you want to be sure it is backed up with complete manuals, drawings and a factory support team that cares.

Processor Technology offers the most extensive documentation of any small computer manufacturer. And we maintain a patient, competent telephone staff to answer your questions.

7. Are maintenance and service people accessible?

Where are they located?

Processor Technology has maintenance and service people in over 50 cities around the U.S.

As you continue turning the pages, see how we stack up to the other computers in this magazine. If we've succeeded in whetting your appetite, see your Sol dealer or write for information on the complete family of Sol computers.

Processor Technology Corporation, Box O,
7100 Johnson Industrial Drive, Pleasanton, CA 94566.
(415) 829-2600.

---

*Available soon.

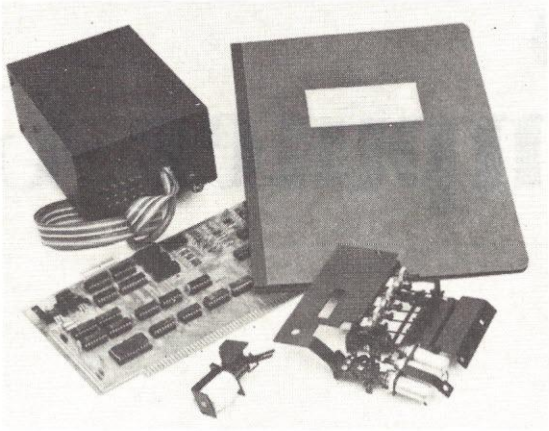
Processor Technology

Your Sol dealer has it.

AZ: Tempe: Byte Shop, 813 N. Scottsdale, (602)894-1129; Phoenix: Byte Shop, 12654 N. 28th, (602)942-7300; Tucson: Byte Shop, 2612 E. Broadway, (602)327-4579. **CA:** Berkeley: Byte Shop, 1514 University, (415)845-6366; Citrus Heights: Byte Shop, 6041 Greenback, (916) 961-2983; Costa Mesa: Computer Center, 1913 Harbor, (714)646-0221; Hayward: Byte Shop, 1122 "B" St., (415)537-2983; Lawndale: Byte Shop, 16508 Hawthorne, (213)371-2421; Orange: Computer Mart, 633-B W. Katella, (714) 633-1222; Pasadena: Byte Shop, 496 S. Lake, (213)684-3311; Sacramento: Micro-Computer Application Systems, 2322 Capitol, (916) 443-4944; San Francisco: Byte Shop, 321 Pacific, (415)421-8686; San Jose: Byte Shop, 2626 Union, (408)377-4685; San Rafael: Byte Shop, 509 Francisco, (415)457-9311; Tarzana: Byte Shop, 18423 Ventura, (213)343-3919; Walnut Creek: Byte Shop, 2989 N. Main, (415)933-6252. **CO:** Boulder: Byte Shop, 3101 Walnut, (303) 449-6233. **FL:** Ft. Lauderdale: Byte Shop, 1044 E. Oakland Pk., (305)561-2983; Miami: Byte Shop, 7825 Bird, (305)264-2983; Tampa: Microcomputer Systems, 144 So. Dale Mabry, (813)879-4301. **GA:** Atlanta: Computer Mart, 5091-B Buford, (404)455-0647. **IL:** Champaign: Computer Co., 318 N. Neil, (217)359-5883; Numbers Racket, 623½ S. Wright, (217)352-5435; Evanston: itty bitty machine co, 1322 Chicago, (312)328-6800; Lombard: itty bitty machine co, 42 W. Roosevelt, (312)620-5808. **IN:** Bloomington: Data Domain, 406 S. College, (812) 334-3607; Indianapolis: Data Domain, 7027 N. Michigan, (317)251-3139. **IA:** Davenport: Computer Store, 4128 Brady, (319)386-3330. **KY:** Louisville: Data Domain, 3028 Hunsinger, (502)456-5242. **MI:** Ann Arbor: Computer Store, 310 E. Washington, (313)995-7616; Troy: General Computer Store, 2011 Livernois, (313) 362-0022. **MN:** Minneapolis: Computer Depot, 3515 W. 70th, (612)927-5601. **NJ:** Hoboken: Computer Works, 20 Hudson Pl., (201)420-1644; Iselin: Computer Mart, 501 Rt. 27, (201)283-0600. **NY:** New York: Computer Mart, 118 Madison, (212)686-7923; White Plains: Computer Corner, 200 Hamilton, (914)949-3282. **NC:** Raleigh: ROMs 'N' RAMs, Crabtree Valley Mall, (919) 781-0003. **OH:** Columbus: Byte Shop, 2432 Chester, (614)486-7761; Dayton: Computer Mart, 2665 S. Dixie, (513)296-1248. **OR:** Beaverton: Byte Shop, 3482 SW Cedar Hills, (503)644-2686; Eugene: Real Oregon Computer Co., 205 W. 10th, (503)484-1040; Portland: Byte Shop, 2033 SW 4th Ave., (503)223-3496. **RI:** Warwick: Computer Power, M24 Airport Mall, 1800 Post Rd., (401)738-4477. **SC:** Columbia: Byte Shop, 2018 Green, (803)771-7824. **TN:** Kingsport: Microproducts & Systems, 2307 E. Center, (615)245-8081. **TX:** Arlington: Computer Port, 926 N. Collins, (817)469-1502; Houston: Computertex, 2300 Richmond, (713)526-3456; Interactive Computers, 7646½ Dashwood, (713)772-5257; Lubbock: Neighborhood Computer Store, 4902-34th St., (806)797-1468; Richardson: Micro Store, 634 So. Central Expwy., (214)231-1096. **VA:** McLean: Computer Systems Store, 1984 Chain Bridge, (703)821-8333; Virginia Beach: Home Computer Center, 2927 Va. Beach Blvd., (804)340-1977. **WA:** Bellevue: Byte Shop, 14701 NE 20th, (206)746-0651; Seattle: Retail Computer Store, 410 NE 72nd, (206)524-4101. **WI:** Madison: Computer Store, 1863 Monroe, (608)255-5552; Milwaukee: Computer Store, 6916 W. North, (414)259-9140. **D.C.:** Georgetown Computer Store, 3286 M St. NW, (203)362-2127. **CANADA:** Ottawa, Ont: Trintronics, 160 Elgin, (613)236-7767; Toronto, Ont: Computer Mart, 1543 Bayview, (416) 484-9708; First Canadian Computer Store, 44 Eglinton Ave. W., (416)482-8080; Computer Place, 186 Queen St. W., (416)598-0262; Vancouver, B.C.: Basic Computer Group, 1438 E. 8th, (604)736-7474; Pacific Computer Store, 4509 Rupert, (604)438-3282.

Processor Technology

Run On Micros Run On



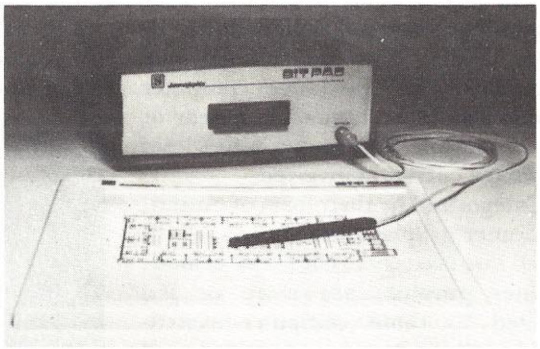
YOUR
MICRO
AND
IBM

Plain-paper hard copy can be as hard to come by as its name if you're not willing to spend just about as much money on a printer as you did on your whole system. Unless you happen to have an IBM Selectric around, one that you could interface to your computer—if there were an easy way to do it.

Well, now there is. ESCON has just introduced a conversion kit. It includes an interface card, power supply, and driver, as well as the cables and all mechanical parts to convert a Selectric typewriter into a printer in a Saturday afternoon.

Fitting all Selectric models, the S-100 compatible kit can be installed without drilling holes or cutting metal parts. And you still have an operating normal typewriter in case your editing system crashes and you're back to hunt-and-peck composing. Complete for $455 from

ESCON
171 Mayhew Way, Suite 204
Pleasant Hill, CA 94523
415-935-4590



SCRIBBLE
SCRIBBLE
SCRIBBLE
SCRIBBLE
SCRIBBLE
SCRIBBLE
SCRIBBLE
SCRIBBLE

This year will be the year of graphics in microcomputing. Starting things off with a bang is Summagraphics' new Bit Pad, a flexible input peripheral for small systems users. Capable of high-quality digitizing, it permits easy entry of positional information as a collection of X, Y values on a compact eleven- by eleven-inch drawing board.

Based on a byte-oriented eight-bit parallel output, the Bit Pad ($555 assembled) will easily interface to almost any micro. Computer animation, real estate planning, design work, opinion sampling, visually coded inventory for the unskilled, etc.—it's all just an electronic scribble away.

Summagraphics Corp.
35 Brentwood Avenue
Fairfield, CT 06430
203-384-1344

A Beginner's Guide to Computer Graphics

or Raster

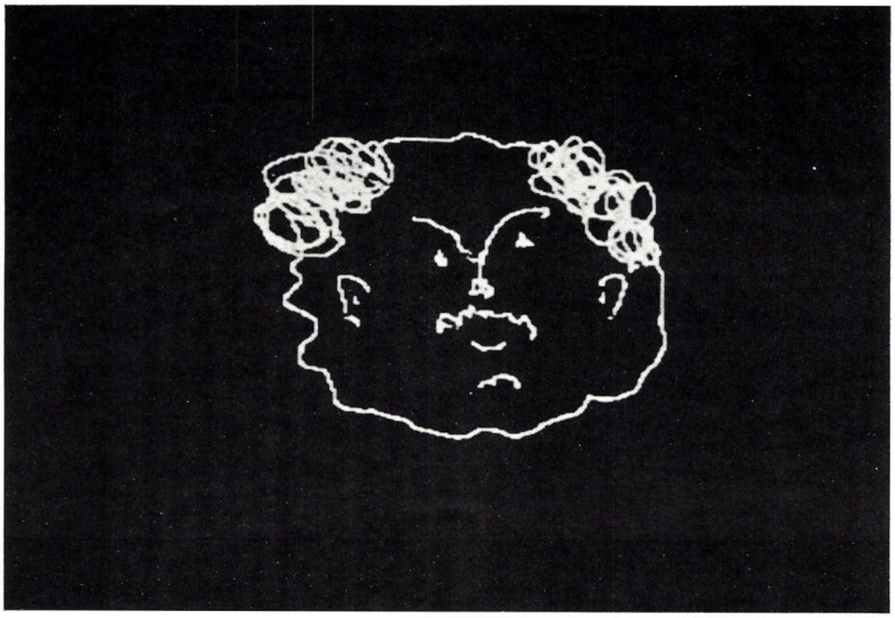
Up until now, the one word which best summed up computer graphics was "expensive." That's not about to change overnight. However, along with much else in the world of computers, graphics systems are beginning to descend rapidly down the price ladder. Low-cost full graphics capabilities for the home computer system should be around in a year or so. Meanwhile, let's look a little closer at what's currently involved.

Graphics systems are beginning to descend rapidly down the price ladder.

Categorized by their form of output, computer graphics can loosely be divided into two types: CRT and printer. Printer graphics are more or less limited to those computer-printed-looking T-shirts and posters we've all seen. Since the CRT form of graphics is not only more flexible, but less expensive as well, it is the form usually associated with personal computing systems. So let's limit our considerations to CRT graphics for the moment. Their mainstay, the cathode ray tube, can further be classified into three distinct types. One is the x-y deflection CRT, the second is the dot-producing CRT, and the third is the raster-scanning CRT. All of these areas overlap to some extent.

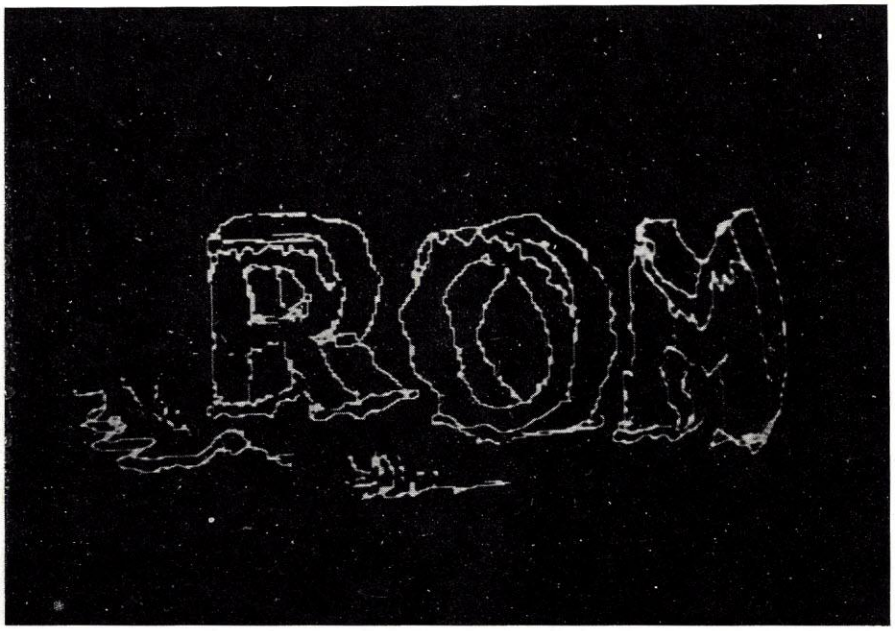
In the field of x-y drawing CRTs, the most popular terminal for graphics is the Tektronix 4010, 4012, 4013, and now the "home computer model" which costs around $7,000—the 4051. See what I mean about "expensive"?

The basic structure of x-y graphics are straight lines called "vectors" which connect two end points. These end points are located by their x and y coordinates (the standard Cartesian coordinates of high school math)—these positions are defined on the tube face in horizontal and vertical directions. Since these positions are specified by digital numbers having only so many possible values (the number of which is determined by the length of the digital number), these systems have a limit of resolution. Any attempted movement of the x-y coordinates of less than this resolution results in no movement at all. The Tektronix 4051 has a resolution of 1024 steps vertically (y axis) and 784



Scan Can

by Bill Etra



horizontally (x axis), although you write programs as if there were 1024 steps on both axes and you just can't see some of the horizontal axis.

In some expensive systems a vector can be a curved line whose curvature is specified from the computer and which is drawn in detail by the hardware in the display. In all vector systems a curved line is made up of small straight lines tacked together. If the computer has to calculate all of these lines it can be quite a drain on the computer's free time. A circle, for instance, takes a lot of computation to display in vector form, which is bad enough if you have to do it only once. If your computer has to do it continuously, fast enough to avoid flicker, display of the circle can be a full-time job for it.

The Tektronix display takes advantage of the "retentive" nature of the luminous phosphor coating on the tube face to display its graphics. Like a standard television tube, the phosphor glows (green in this case) when struck by the electron beam generated in the "neck" of the tube. Unlike TV picture tubes, the phosphor keeps on glowing green for a very long time thereafter, until the image is "erased" by a signal from the outside equipment. With this kind of "storage tube," an image can be "painted" on the face of the display by the movements of the electron beam. It does not have to be stored elsewhere in memory, which at the time of the storage tube's development was frightfully expensive.

In other types of x-y displays, called refresh vector graphics, the beam's image is not stored by the surface of the screen. Instead, the beam is continually scanned as a line drawing of vectors. This is done by having a memory buffer, which reads out at a continuous rate a certain number of x-y coordinates in the x-y deflection system of a cathode ray tube. The picture is continuously refreshed—essentially redrawn many times each second. This requires not only a special-purpose cathode ray tube, but large amounts of computer memory as well, in which to store the huge number of vectors.

Basically, then, we have two forms of display etched by the electron beam onto the video screen: the semipermanent storage phosphor display and the refresh display.

Vector graphics systems are good for drawing outlines. They're not good for drawing solid bodies, because a solid body must be shaded by hundreds and thousands of small line segments requiring more computing power than that with which the system can deal efficiently.

Dot graphics systems just put the light out at single points. These points are also specified along Cartesian coordinates—x and y. There's an advantage to dot systems. Everything's the same size. All the lines are the same size—one unit. But since it works on x-y coordinates in the manner of vector systems, it has the same limitations. In order to draw a circle, for instance, you must specify a great number of very small dots. Another unfortunate quality of dot graphics systems is that everything looks as if it has been punched out of cut paper. Look at some graphics from the PDP-11 DECdot drawing system some time, if you get a chance, and you'll see what I mean.

An unfortunate quality of dot graphics systems is that everything looks as if it has been punched out of cut paper.

A system which is becoming the more popular all the time is raster graphics. Raster graphics work like TV. The x-y position of the beam advances continuously; only whether the beam is on or off at any one point as it sweeps horizontally

Pictures executed by A. D. Libin using the Etra "Video-Sketch" program

across the screen is controlled. You have a column and a row counter and you read out so many dots on one line, move down a fraction, write the next line bit by bit, and so on, until you get to the bottom, shut off, and go back to the top to start over. A good number of raster-scanning graphics systems work at exactly the same speed and line count as your TV.

Others, especially in the home-computer market, such as the Compucolor and Apple machines read out just like TV but at their own internal timing. TV has 525 lines in a frame; it puts out half a frame, or a field, every sixtieth of a second. These two sets of lines, or frames, are displayed alternately—as if you interlaced your fingers—and it is in fact referred to as interlace. Your eye is what puts them together, so you see one full frame every thirtieth of a second. The Compucolor system handles its color in a slightly different manner from home color TV, so its signals can't be displayed that way. But a more detailed description of the workings of video display, particularly color video display, belongs in another article.

There are other scan systems; the timing is what tends to change. For instance, the Hewlett-Packard terminal scans 800 lines, black and white, returns to the top of the screen, and scans another frame. The advantage of scanning more lines is obviously higher resolution; the disadvantage is that it is not compatible with a normal TV set. In the case of Hewlett-Packard's new system it also doesn't have a three-by-four format. All TV pictures appear as three units by four units: three units in the vertical plane, four units in the horizontal plane. Since this follows the rules of composition we're used to in dealing with western civilization, other formats become somewhat harder to deal with, as the movie people discovered when they started to do super wide-screen projections.

Raster graphics is quickly becoming the most important type of video output.

Raster graphics systems depend on memory. You enter into both raster graphics systems and refresh graphics systems the same way: you put information into memory and it gets read out as an illuminated position on the screen. The raster system reads out through a special timing which gives it columns and rows. Here again we run into problems. (Problems seem to be the underlying unity of graphics systems.) One of the main difficulties in raster systems is that you need 512 dots by 512 dots by 8 bits minimum to produce a single video frame. The 8-bit-deep point gives you 256 colors or gray values. That comes out to 512 times 512 which is roughly a quarter of a million bytes of information in order to store one video frame. This is one of the reasons why the raster scan type devices, the Apple computer and the Compucolor, for instance, produce images that don't look like real television frames. Even so, raster graphics is quickly becoming the most important type of video output.

Raster scan graphics can be addressed in several ways. You can either get an x-y address which puts it into the proper dot on the buffer, or you can address using a memory map, which is the principle behind Cromemco's TV Dazzler. With such a map you don't give x-y coordinates, but store the color data in an actual space in memory. It's left to your program to figure out where you put things. But all these are raster scan devices. The Matrox has real TV timing, the Dazzler has sixty-two-cycle vertical sync so it's slightly off from real TV timing, and I don't know about Apple's machine. I haven't had a chance to look at it that carefully.

There are two other types of raster scanning. One is alphanumeric graphics and the other is programmable-character-set graphics. Basically, on something like the Sol terminal, you have sixty-four character positions and sixteen lines down. In those positions, by combining characters and by switching between characters on alternate fields of video, you can create graphics. That's how the Processor Technology *Target* game is achieved. That's also how their *Star Trek* symbol is created. There is a surprising amount of graphics that can be executed using an ASCII character set which has 96 to 128 usable characters.

There is yet another way. Instead of bringing the dot patterns which form your characters from an ASCII look-up ROM, you can have the program lay out the patterns of dots within each character. This is programmable-character-set graphics and is what ECD has advertised that they are doing in their machine, Micromind. The potential is there, though I haven't seen it working really well yet. The advantage of programmable character sets is that they allow you to change the actual characters. You can build new graphic characters that in turn go together in building blocks.

It is a good way of getting high resolution with relatively little memory. Because the characters will be twelve by eight units or five by seven units or whatever, they'll have only that many dots in them, thus allowing the use of very small buffers. Larger buffers will, of course, give you a wider range of characters from which to choose.

Programmable-character-set graphics is especially good for things like interactive games. The problem with programmable-character-set graphics is that you only have a limited number of spaces for the characters, usually somewhere under one hundred in one dimension, under fifty in the other. And though you can put high-resolution characters in there, you have to move them incrementally from one character space to another. They have to appear and disappear or change within that space. So, over all, it doesn't really give you very high resolution, not compared to the 512 by 512 lines you need to recreate a real television picture.

But this whole thing is changing. And it will change even more drastically as memory becomes cheaper and cheaper. You'll get bigger and bigger frame buffer-type graphic instruments and we can expect to see within the next ten or twenty years people at home who have systems capable of storing single or even multiple frames of video in color. Certainly, if bubble memory was fast enough, which it doesn't appear to be in the commercial versions at the moment, it would permit a large step forward in computer graphics.

Meanwhile, computer graphics offer more possibilities right now than most people realize. For starters, you can turn your video monitor into an electronic drawing board. ▼

YOUR COMPUTERIZED DRAWING BOARD

This program will turn any 8080 computer using an S-100 bus into an electronic "Etch-a-Sketch." Your system will need a joystick and a 7A+DI/O (strapped according to instructions) as well as a Matrox video buffer. It will then draw lines on the video screen corresponding to the joystick's position.

To remove unwanted joystick motion from the joystick operation and the curves, it is suggested that a 47-microfarad capacitor be placed between both x and y joystick wipers and a ground, one end of the potentiometer. Make sure that the polarity of the voltage across the capacitor matches the polarity of the capacitor.

Pressing button number one on the joystick lifts the pen off the finished drawing; number four turns on a blinking dot. Buttons two and three clear the screen to white and black respectively.

To begin drawing, you enter a dot. Pressing numbers one and two simultaneously will load a black dot. Pressing numbers two and three simultaneously will load a white dot. To change dot colors, press three and four at the same time. This will reloop the program, and you start over pressing either buttons one and two together, or two and three together.

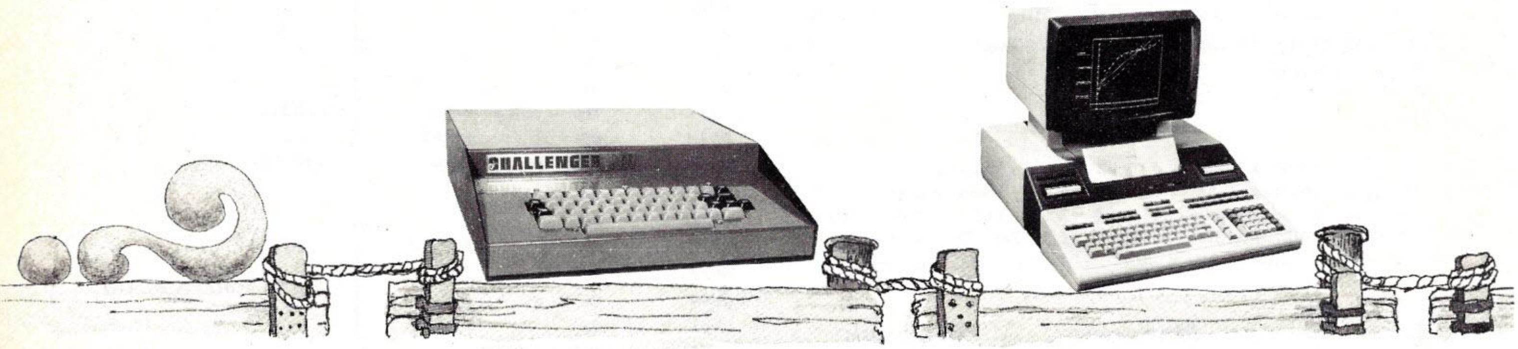
Device Mnemonics

18 = JSB — JoyStick Buttons
19 = JSX — JoyStick X-axis
1A = JSY — JoyStick Y-axis
00 = VBD — Video Buffer Dot in
01 = VBX — Video Buffer X-axis
02 = VBY — Video Buffer Y-axis
03 = VBE — Video Buffer Erase
00 = VBS — Video Buffer Status out
0D — Zero Decimal
1 DEC — One DECimal

```
Address   Machine   Mnemonics       Comments
in Hex    Code

0000      DB        IN
0001      18        [JSB]
0002      FE        CPI
0003      FC        [JSB#1 + #2]
0004      CA        JZ
0005      0E                        LOOP LOOKING FOR
0006      00                        SET DOT WHITE
0007      DB        IN              BUTTONS 3 AND 4 OR
0008      18        [JSB]           SET DOT BLACK
0009      FE        CPI             BUTTONS 1 AND 2
000A      F9        [JSB#2 + #3]
000B      C2        JNZ
000C      00
000D      00
000E      4F        MOVC,A
000F      06        MVI,B           SET OFFSET FROM ± 28
0010      80        128D            TO 0 TO 255
0011      DB        IN
0012      19        [JSX]
0013      80        ADD,B           SET X LOCATION
0014      D3        OUT
0015      01        [VBX]
0016      DB        IN
0017      1A        [JSY]
0018      80        ADD,B
0019      2F        CMA             SET Y LOCATION
001A      D3        OUT
001B      02        [VBY]
001C      DB        IN
001D      18        [JSB]
001E      FE        CPI
001F      FE        [JSB#1]         CHECK FOR
0020      CA        JZ              PEN LIFT BUTTON #1
0021      0F
0022      00
0023      79        MOVA,C
0024      D3        OUT             OUTPUT DOT
0025      00        [VBD]
0026      DB        IN
0027      18        [JSB]
0028      FE        CPI
0029      FD        [JSB#2]         CHECK FOR CLEAR
002A      CA        JZ              SCREEN WHITE
002B      37
002C      00
002D      DB        IN
002E      18        [JSB]
002F      FE        CPI
0030      FB        [JSB#3]
0031      CA        JZ              CHECK FOR CLEAR
0032      3E                        SCREEN BLACK
0033      00
0034      C3        JMP
0035      4B
0036      00
0037      3E        MVI,A
0038      01        1D
0039      D3        OUT             CLEAR
003A      03        [VBE]           SCREEN
003B      C3        JMP             WHITE
003C      42
003D      00
003E      3E        MVI,A
003F      00        [0D]
0040      D3        OUT             CLEAR SCREEN BLACK
0041      03        [VBE]
0042      DB        IN
0043      00        [VBS]
0044      E6        ANI
0045      01        [1 DEC]         CHECK TO MAKE
0046      FE        CPI             SURE SCREEN IS
0047      00                        CLEAR BEFORE
0048      C2        JNZ             GOING ON
0049      42
004A      00
004B      DB        IN
004C      18        [JSB]
004D      FE        CPI             CHECK FOR BUTTON
004E      F7        [JSB#4]         #4 CURSOR MODE
004F      CA        JZ
0050      5C
0051      00
0052      DB        IN
0053      18        [JSB]
0054      FE        CPI
0055      F3        [JSB#3 + #4]    CHECK FOR DOT
0056      CA        JZ              RESET BUTTONS #3
0057      00                        AND #4
0058      00
0059      C3        JMP
005A      0F
005B      00
005C      79        MOVA,C          OUTPUT
005D      2F        CMA             COMPLIMENTARY
005E      D3        OUT             DOT FOR CURSOR
005F      00        [VBD]
0060      C3        JMP             LOOP TO NEXT X
0061      0F                        NEXT Y
0062      00
```

HOME COMPUTERS
A Look at

by Jules H. Gilder



What started out as a few hundred thousand dollar hobby market almost three years ago is about to explode into a giant consumer market with a total sales of almost two billion dollars by 1985. The growth of the personal-computer market—surprisingly large in the last few years—is expected to increase exponentially with the consumerization of the computer. And that consumerization is taking place right now as numbers of user-oriented computers become available.

The first consumer computer to be announced was the PET (Personal Electronic Transactor), from Commodore Business Machines in Palo Alto, California. It was the first hobby computer designed with the consumer in mind. Not a bag of parts to be assembled, as the earliest home computers were, and not an assembled board to be connected to interface circuitry, a power supply, a cabinet, and a terminal, as later home computers were, the PET is a complete off-the-shelf home-computing system that comes out of the box, plugs into the wall, and is ready to use. It contains a keyboard, CRT, and cassette tape storage. And it costs only $595 for the 4K RAM model, $795 for one with 8K RAM.

A stand-alone computer, the PET consists of a nine-inch black and white CRT that can display up to a thousand characters (twenty-five lines by forty columns), an ASCII keyboard, and a numeric pad, as well as a cassette recorder for program storage. The PET also contains a memory expansion bus, a port for an additional cassette recorder, a user-controllable eight-bit bidirectional parallel port, and an IEEE 488 interface port.

The PET contains BASIC language in 8K of ROM, and it's ready to program in BASIC the moment the power is on. When first turned on, the computer determines how much memory is available for user programs and displays the result on the CRT screen.

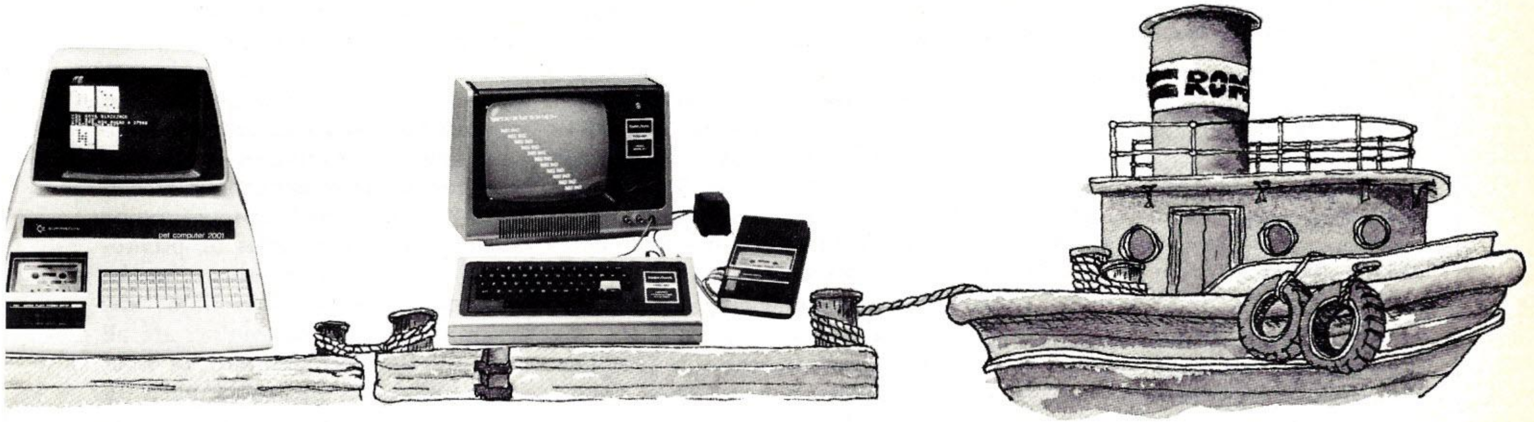
The PET is designed to be easily serviced by TV repairmen. It contains three printed-circuit boards—one for the cassette recorder, one for the CRT monitor, and the main computer board. The last will be the one least familiar to TV servicemen, so Commodore has built into it a self-diagnosis feature which tells the serviceman whether the board is functioning properly or not. If it isn't, it can easily be snapped out and replaced.

The PET, because it was the first consumer computer introduced, naturally set the pace for its competition. Other manufacturers quickly sat down at their drawing boards to see if they too could come up with a computer for the average citizen.

Shortly after the Commodore announcement, Radio Shack, first to follow suit, divulged that it, too,

Illustrated by Cindy Hain

What's Coming



had a put-together home computer.

Radio Shack's TRS-80 costs $599.95. Like the PET, the TRS-80 has a keyboard, CRT display, and a cassette tape memory. But unlike the PET, which is built as one complete unit, the TRS-80 is modular; and, if one wishes, the computer can be purchased without the CRT and tape recorder for only $399.

The PET has a 6502 microprocessor for a CPU; the TRS-80 uses a Z-80 microprocessor. And although the Z-80 is generally considered a more powerful micro, the performance of the PET exceeds that of the TRS-80—at least for now (it remains to be seen what will happen once Radio Shack gets its Level II BASIC online).

Radio Shack's computer comes with 4K each of ROM and RAM. The ROM can be internally expanded to as much as 12K, and as much as 16K of RAM can be held in the computer before an expansion add-on is needed. The CRT display for the TRS-80 is a twelve-inch display that holds sixteen lines of sixty-four characters each.

Another entry in the home-computer race is being tendered by Ohio Scientific. Their new $598 computer, the Challenger IIP, is functionally similar to the PET—but with some significant differences. To begin with, it is capable of handling color graphics, while the PET works only with black and white. However, the Challenger IIP does not come with a CRT display. Instead, it produces a video signal that can be fed by means of an RF modulator to the antenna terminals of a color television set.

Ohio Scientific is offering another new computer—you guessed it—the Challenger III. What's unique about this unit? It contains three micros: a 6800, a 6502, and a Z-80; so it can run virtually all the software that is published. The Challenger IIP is designed for the consumer; Challenger III is

ROMtutorial ROMtutorial

Interface: The boundary between different sections of a computer, or between the computer and the outside world. The circuitry crossing the boundary is also called the interface.

Keyboard: A device like a typewriter keyboard that enables a person to key information into a computer. The information, in sequences of characters (letters, digits, and special symbols), is called text strings, or simply text.

Cassette tape storage: Typically, a personal computer has memory, or storage, for a fairly limited number of data items in its fastest random-access memory (RAM). Frequently memory space is needed for a large amount of data. A good place to put the data is often a simple cassette tape, since it holds a lot and is cheap. There are a variety of methods for adapting normal cassette recorders to work with computers, but most of them convert the on-off signals of the computer into tones that can be recorded easily.

ROMtutorial ROMtutorial

CRT: Cathode ray tube. A television-style display device.

K: From kilo, which is in turn from the Greek *khilioi,* meaning a thousand. In computerese, however, K usually means 1024 (2^{10}) instead of 1000. Quantities expressed in binary, like memory addresses, are commonly figured in batches of 1024. For instance, 4K is equal to 4096.

RAM (random-access memory): Memory like a set of pigeonholes into any of which the computer can put new information or from any of which it can read old information. The computer can choose any pigeonhole (or address) at any time. RAM can store and recall information at high speeds, and information stored in it can be changed at any time.

ASCII: Acronym for American Standard Code for Information Interchange. A standard code used to represent letters and numbers in binary on-off form. Seven distinct binary units, or bits, are used for each character; that way a total of 128 different combinations can be produced.

Bus: A bus is a wire connected to many places. Normally, it's used to carry electricity for power, but in a computer, very fast, low-power electrical impulses are sent between sections on buses. In order for a lot of these signals to happen at once, computer buses have a lot of wires in them.

Bit (binary digit): The kind of number computers use, 1 and 0, also equivalent to an on or off condition.

Bidirectional parallel ports: A port is some arrangement for getting data signals into or out of a computer. A bidirectional port is one that allows data to go into *and* out of the computer. A parallel port is one that has the signals running in parallel—meaning that a number of signals travel through the port at once. (The alternative is a serial port, through which the signals travel one after the other.)

designed for the more experienced user, and is meant to compete with up and running hobbyist and small business systems in the $2,000-$3,000 price range.

Even with its added processing power, Challenger III costs only ten percent more than these conventional single-processor computers. For example, a unit with 32K of semiconductor memory, a storage capacity of 500K bytes on dual floppy disks, and a serial interface costs $3481. For applications where enormous amounts of data must be stored, Ohio Scientific offers an optional seventy-four-megabyte hard disk system for under $6,000.

The software available with all three computers includes BASIC, but PET and Challenger BASIC are more powerful than TRS-80 BASIC. Radio Shack is working on an improved EXTENDED BASIC that will undoubtedly bring the TRS-80 up to par with the other two, but it may take a while, and it will probably cost extra.

Other manufacturers are jumping onto the consumer-computer bandwagon. MITS, which started the whole thing three years ago when it introduced the Altair 8800, doesn't want to be cut out of the looming consumer market and is playing with a new no-kit machine in its research labs. Also rumored to be under development at MITS, according to industry sources, is a machine compatible, not with the S-100 bus, but with Intel's SBC bus instead. The official word from MITS, however, is that this is not so.

A low-cost competitor is entering the fray from RCA. Known as the COSMAC VIP, it will cost only $275. RCA is billing this computer as a viable alternative to the expensive BASIC computers. Interfacing to a standard black and white TV set, it is aimed primarily at the games and video-graphics markets. A color conversion unit for the VIP will be available in the near future.

The heart of the VIP is RCA's own CDP 1802 microprocessor—that was to be expected. Unlike the other new entries in the home-computing field, however, RCA's is available only as a kit. The $275 gets you the CPU, 2K bytes of RAM, all the necessary support circuitry, a new single-chip graphic video display interface IC (the CDP 1861), a hex keyboard, and an 8½-by-11 printed-circuit board. Onboard RAM can be expanded to 4K. The VIP also features a 100-byte per second audio cassette interface, a special sound-effects circuit, and a built-in power supply. The unit is expandable by means of two connectors: one is for plugging in extra memory; the other is for expanding the input/output capability of the machine.

Even for those deathly afraid of machine-language programming there is hope.

One drawback of the VIP may prove to be the need to program it in machine language. RCA is trying to reduce the negative connotations often associated with machine-language programming by providing an interpretive programming language called CHIP 8. CHIP 8 is not so much a programming language as it is a series of preprogrammed subroutines that can be incorporated very easily into any program—and the subroutines are all related to the video graphics. In other words, RCA has preprogrammed all the difficult things so that even a novice can write his own game programs. Even for those deathly afraid of machine-language programming there is hope. RCA is going to include in the kit, just to begin with, the program listings for twenty different video games that anyone can enter into his computer and run. More programs will undoubtedly follow.

Other manufacturers are also looking at TV games as the way to enter the home-computing market. Atari is said to be working on a programmable unit featuring color graphics; it will use either custom chips or a 6502 micro. Bally, the pinball-machine manufacturer, is coming out with a Z-80-based machine that includes some of the custom chips Bally originally developed for its pinball market.

The Bally Library Computer, as the new pinball-spawned machine is called, sells complete and ready to use, except for the standard TV receiver, for $299.95. Well, actually, that's the basic, physical shelves—without the library. The basic computer con-

THE PERSONAL COMPUTER MARKET

(Figures in Millions of Dollars)

	1976	1978	1980	1982	1985
Professional	105	140	200	330	800
Hobby	38	120	187	185	100
Consumer		100	385	640	1000
Total	143	360	772	1155	1900

tains 12K bytes of memory. By plugging in different cartridges—tally up the Bally set at $24.95 apiece—it is possible to run many a program. The cartridges have up to 8K bytes of additional memory, containing the desired programs.

The Bally unit comes initially with a calculator-type keyboard for data entry. A typewriter keyboard will soon be available, as will dual magnetic tape decks and a high-speed printer. These accessories will make it possible to program the machine in BASIC.

A new entry in the desktop-computer market which would be ideal for hobbyists if it weren't for its $12,000 price is the Series 9800 System 45 from Hewlett-Packard. It features a twelve-inch CRT display, a BASIC interpreter, applications software, and a built-in high-speed printer (optional). In addition, it can be ordered with dual built-in tape decks. The System 45 has multiple interface ports and the capability to use four standard interface types: bit-serial, binary-coded decimal, bit-parallel, and IEEE 488.

It is interesting to note that the basic $12,000 unit from Hewlett-Packard contains 16K bytes of RAM, the CRT display, and one tape transport; the PET (which also has an IEEE 488 interface) contains in its deluxe model, selling for $800, 8K of RAM, the CRT display, and two tape transports. What does HP give you for the additional $11,000 and then some? Meanwhile, back to some lower price tags.

No one seems to have a good handle on what TI has up its sleeve.

As with most consumer products these days, we can expect the Japanese to make a strong push in the home-computing market. The first entry will most likely be a Hitachi 6800-based machine. It should cost about $600, and it can be expected by the middle of this year.

National Semiconductor is rumored to be working on a low-cost entry that will use an 8080 micro and be similar to Commodore's PET. But despite the fact that National has just created a new Personal Computer Components group, the company denies the PET-competitive rumor.

Finally, the eyes of industry insiders are on Texas: they're all anxiously waiting to see what Texas Instruments comes out with. Most of them remember how TI managed to take over a major share of the calculator market with its aggressive pricing, but no one seems to have a good handle on what TI has up its current sleeve. Some think they'll come out with something that looks a lot like their Silent 700 terminal but has a floppy disk on it. Such a device might be out by the end of the first quarter of the year. Other TI watchers feel that it will be late spring before TI comes out with a consumer computer, and that when it arrives it will look a lot like a PET. Still others believe that even by fall all we'll have from TI will be strong hints about a new machine that's going to give everyone else stiff competition. This machine, equipped with bubble memory, might debut next year.

Detractors of this last view note that bubbles are still too expensive and too slow to appear in home systems for at least another four years. But it should be pointed out that TI is already making a Silent 700 series terminal with bubble memory and that 20K byte modules of bubble memory are available from the Texas semiconductor company for only $500. Two years from now that price could be drastically cut.

A survey of manufacturers of home computing equipment reveals several things that can be expected to happen in 1978. These include:

- Cheaper CRT terminals with more capability

ROMtutorial ROMtutorial

IEEE: Institute of Electrical and Electronics Engineers. An organization of technical people.

BASIC: Acronym for Beginner's All-purpose Symbolic Instruction Code. An easy-to-learn, easy-to-use programming language especially adapted for use with mini- and microcomputers, as well as time-sharing systems. It provides anyone using the computer with instantaneous feedback on whether he's doing all right or making a mistake.

ROM (read-only memory): Memory, or storage, whose contents are fixed at the time the information is entered. It's like a telephone directory the computer can read, but not scribble in. ROM is used to hold instructions (programs) for the computer.

Printed circuit (PC) board: A thin sheet of insulating material to which a copper foil has been bonded; by a process similar to photolithography, the copper foil is selectively etched away so as to leave thin strips of foil, called traces. Electronic components are mounted on a printed circuit board by inserting their leads (the wires that emerge from the component) into holes drilled through the board. Each hole is positioned so as to pass through a particular trace, and the leads are soldered to the traces. The traces thus serve as wires for interconnecting various components on a printed circuit board.

Microprocessor: The "thinking" section of a computer is called the central processing unit (CPU), or simply the processor. If it's so small that you need a microscope to examine it, it's called a microprocessor.

RF modulator: An RF modulator converts a computer signal to one in the radio frequency range.

Software: The programs run on a computer. A computer system consists of hardware—the computer and its accessories—and software—the programs which make the hardware work the way you want it to.

Byte: A piece of information consisting of eight bits. It has 256 possible values.

ROMtutorial ROMtutorial

Dual floppy disks: A floppy disk (also known as a diskette) is a flexible sheet of plastic with a magnetic coating, used for moderately high speed storage of data or programs for a computer. Although it looks like a 45-rpm record, it acts more like a magnetic tape. A dual floppy is simply two floppy disk drives in one cabinet.

Serial and parallel interfaces: Usually, a wire or cable carrying electrically coded information crosses the dividing line—the interface—between two electronic devices. If the cable carries several different signals at once, it's a parallel interface. If the information moves in a sequence, single file, through a wire, the interface is serial.

Megabyte: One million bytes. A byte is a fundamental unit of storage in a computer; a million is an awful lot.

Hard disk: A very fast storage device for a computer. A hard disk is similar in concept to a floppy disk, but it's usually an order of magnitude bigger and faster. Physically, a hard disk is a platter of lightweight metal with a magnetic coating on it. A drive spins the disk while a magnetic recording head traverses the disk, writing and retrieving data.

S-100 bus: The S-100 is a 100-wire bus used for many personal computers; because they have the same pattern of four interconnections, the plug-in board from one will work (usually) when plugged into another S-100 machine. The S means "standard."

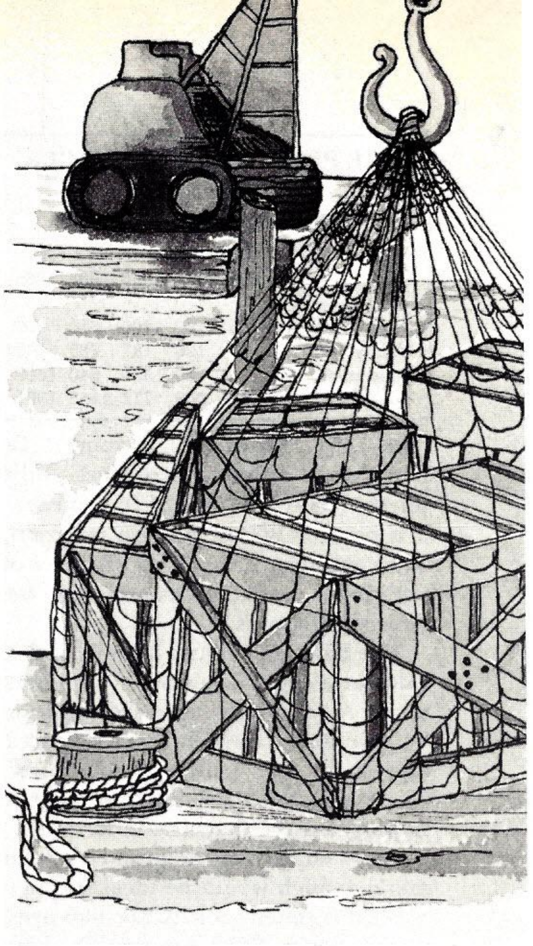
Hex: Hexadecimal. A hexadecimal digit is a member of the set of sixteen digits 0 through 9 and A through F, where A through F represent the decimal numbers 10 through 15.

Audio cassette interface: An interface, or boundary-line, circuit which converts information signals received from a computer into audio tones which can be recorded on a regular cassette tape recorder. It will work in the reverse direction, too; information can be played back into the computer from a cassette.

Machine language: The internal programming language used by a computer. Different computers usually have different machine languages, which may or may not be compatible.

- Improved and exotic peripherals
- Development of software publishing companies
- A shakeout of the small guys selling home computer systems

The development of CRT terminals that cost less and can do more will result largely from the availability of new CRT controller ICs such as the recently announced CRT 5027 from SMC Microsystems Corp., in Hauppauge, New York. SMC's new device self-contains most of the circuitry ordinarily required to build a CRT terminal; it replaces between 30 and 180 generally required integrated circuits (the number of ICs actually replaced depends on the complexity of the terminals).



Terminal costs can be cut by seventy-five percent or more with this new IC. For example, by using the CRT 5027 IC and ten other ICs, it is possible to manufacture a dumb terminal selling for around $200. The current price for the package is $1000. Think how much more could be saved if you built it yourself.

As if the substantial price savings were not enough, the chip makes possible things not generally possible with available terminals, split-screen capability, for instance. Half of a screen could display an original listing while corrections were made on a duplicate on the other half of the screen. And each half of the screen could be scrolled independently.

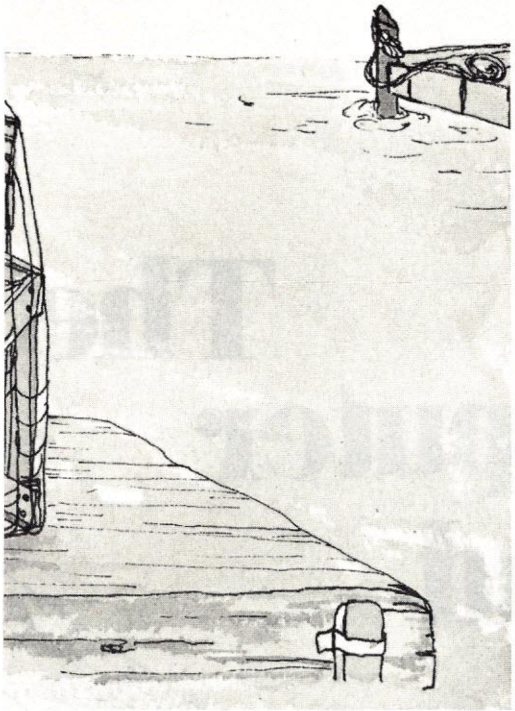
The CRT 5027 also provides electronic wipe capability. And the display format is completely software controlled. This means that a display with a variety of different character heights, widths, and line lengths can be produced. In addition, 132-column-per-line output (standard printer output) is possible.

> *If you are interested in a floppy and don't have an urgent need for one, you might do well to wait a few months for it.*

A big improvement in the type and quality of peripherals available is expected this year. One of the things likely to appear before the year is out is a high-quality speech synthesizer. Industry sources say that Commodore is planning one as an add-on to the PET. Commodore is not commenting. The company does note, however, that it will try to fill what it sees as a big hole in the peripherals area by coming out with a Selectric typewriter interface. And Commodore is working on several other peripherals as well, including a low-cost printer based on the Practical Automation impact printer mechanism and a floppy disk system.

If you're interested in a floppy and don't have an urgent need for one, you might do well to wait a few months for it. Industry sources note that there should be some stiff price competition in the minifloppy area later in the year, and that's good for the hobbyist.

The widespread use of minifloppies will bring with it a large need for floppy-based software. Sophisticated disk-operating systems and improved high-level languages will become available. FORTRAN, which has only now begun to appear in the hobby market,



will become more widely available. There will also be new, improved versions of BASIC. The reason is that this will be the third time some of the software designers are writing BASIC; as with anything else, the more you do it, the better you are at it.

Expect to see competitive software pricing too. It's starting now. Some people are selling FORTRAN for $500—Cromemco is selling FORTRAN for only $95. The catch is, you have to buy their disk system to get it at that price. But if you have to go out and buy a disk system anyway, why not theirs?

> *The future dollar volume of software is expected to be many times larger than that of hardware.*

Systems software is not the only area in which software advances can be expected. Manufacturers are providing ever-increasing support for applications software as well. MITS, for example, has set up a separate subsidiary, the Altair Software Distributing Company, for the sole purpose of obtaining and refining software, then licensing and distributing it.

The future dollar volume of software is expected to be many times larger than that of hardware. The market seems so promising that it looks like there will be quite a few software publishing companies by this time next year. In spite of current experiments in machine-readable code and flexible vinyl records with programs recorded on them, the mass marketing of software will probably settle on the cassette tape as its vehicle. One industry observer notes that the music industry makes money selling cassettes at seven dollars apiece. So the hobby-computer industry should be able to make money on programs at twenty dollars apiece. (The higher price takes into account a lower anticipated volume.) Prices can probably drop further if producers of tape interfaces can come up with a system that permits the use of tapes duplicated on standard audio equipment. The biggest problem with the cassette approach is that, in spite of the industry's best productive efforts, there is no standard tape interface. The ultimate vehicle for conveying programs, therefore, may turn out to be a ROM cartridge similar to that now used in video games.

Look-alike low-cost copies of the IBM 5100 computer are going to surface in numbers this year. The 5100, designed for small-business applications, comes with a keyboard, CRT display, and tape cassette for program storage—all in one box—but a box a little bigger than the one the hobbyist is apt to have opened heretofore. The 5100 operates in both BASIC and APL, and it sells for about $10,000. Competitive units will not be carbon copies of the 5100, but they will have similar capabilities. They will cost between $3000 and $5000, and they will have lots of computing power, probably derived from a sixteen-bit microprocessor. Chips, after all, are getting bigger and better every day, even while they're getting smaller. ▼

ROMtutorial ROMtutorial

Bit-serial: With one bit following another in time sequence. One man shooting twenty-one bullets is an example of a serial operation. Replace the bullets with bits and you have the distinction.

Bit-parallel: With a number of bits transmitted simultaneously. A twenty-one gun salute is an example of a parallel operation.

Binary-coded decimal: Ordinary decimal numbers aren't represented as decimal numbers in a computer. Instead, they must be coded in binary. Binary-coded decimal (BCD) is a way of doing this, representing each decimal digit (0-9) as a binary integer four bits long (0000-1001).

Bubble memory: Integrated circuit memory using "bubbles" of magnetized circuit base material to represent bits of data. The bubbles can be moved around by controlled magnetic fields and do not disappear when the power goes off.

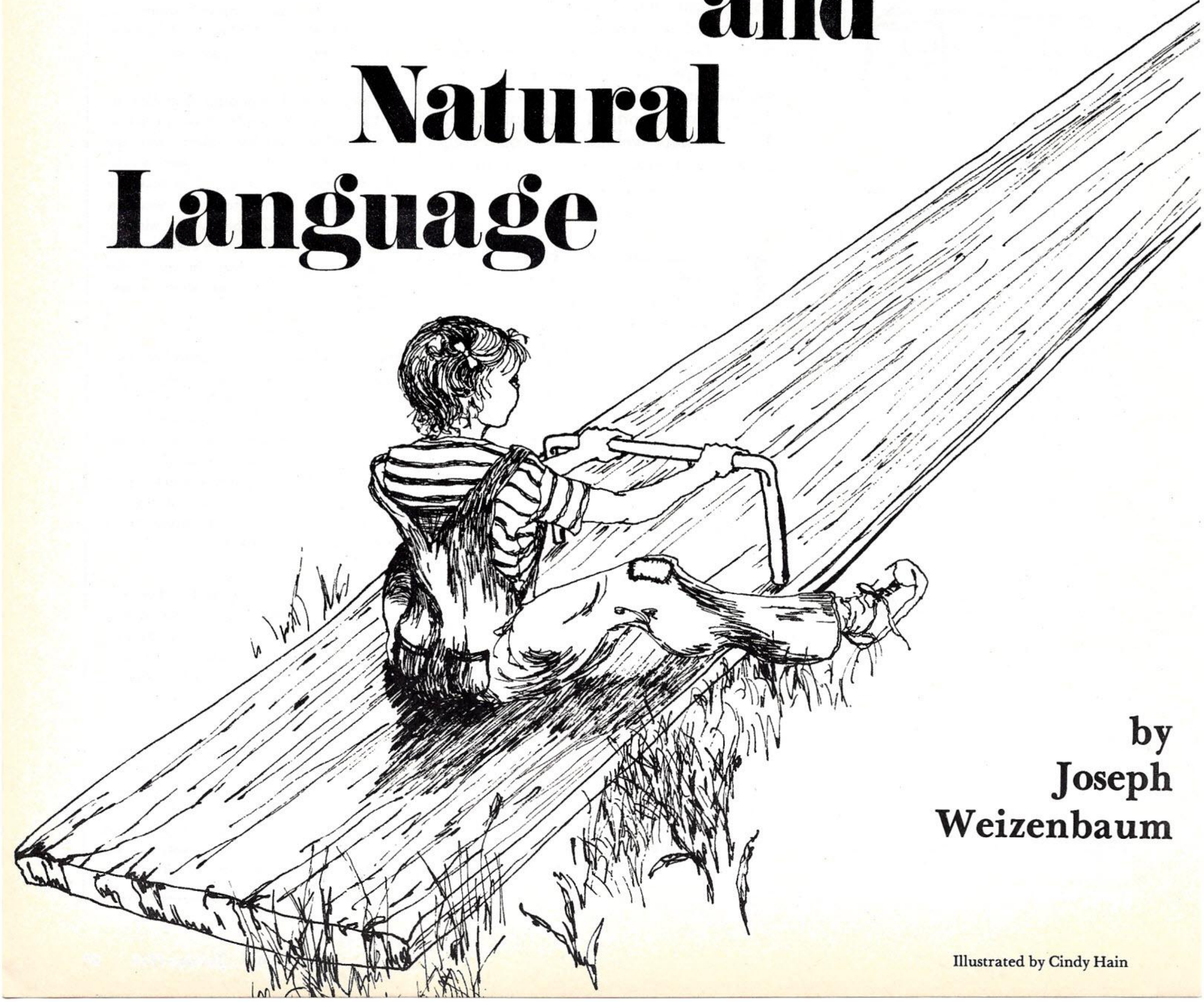
Scrolling: A method of displaying data on a TV screen. The display "rolls up" a line when the line has been filled; you can recall the data as if the screen were a window opening on a long roll of text. By pressing keys on the terminal, you can move the text around behind the window as if it were a scroll. It's a lot easier to read than a "paging" display, in which the entire screenful disappears after the last line is written.

Higher-level language: Programming language oriented toward the problem to be solved or the procedures to be used. Contrasts with machine language, which is the basic language of a computer. Programs written in machine language require no further interpretation by a computer. Usually, however, a higher-level language is easier to program in than a lower-level language.

FORTRAN: An acronym for Formula Translator. FORTRAN is one of the oldest and most widely used programming languages. Much of its use is in scientific applications, since its strength lies in its handling of numeric quantities.

APL: Short for A Programming Language. A mathematically structured programming language developed by Ken Iverson and Adin Falkoff of IBM. It is very "dense"—a great deal can be expressed in very little of it—and very consistent. (See "APLomania" in August 1977 ***ROM***.)

The Computer and Natural Language



by Joseph Weizenbaum

Illustrated by Cindy Hain

In the December issue of *ROM* we distinguished three modes of artificial-intelligence research: the so-called performance, simulation, and theory modes. We observed, however, that the distinctions between them are not absolutely sharp. Moreover, we concluded that "theory" as used in the term "theory mode" has to be taken somewhat less than literally. The use of ideas derived from computers and computation in attempts to understand the human mind is rather more metaphorical than, say, is the use of mechanistic ideas in the understanding of the physical universe. But if we leave aside the vast body of work on modern computer science that deals either with theoretical issues concerning computation itself (e.g., finite automata theory or the theory of the structure of programming languages) or with the direct application of computers to specific tasks, independent of whether the execution of such tasks would count as intelligent behavior if it were accomplished by a human (e.g., the solving of systems of differential equations or the computer control of some complex chemical process), we are left with a subdomain of computer science in which at least one of the major aims is the imitation of man by machine. It will not prove useful for the purposes of this article to emphasize the various ways in which the work in this domain may be assigned primarily to psychology or to linguistics or to whatever established discipline. I shall therefore not press such distinctions in what follows.

Two things are clear: If we wish a machine to do something, we have to tell it to do it, and the machine must be able to understand what we say to it. The most common way to tell a computer what to do, at least to this day, is to give it a specific program for the task we have in mind and, of course, the data to which that program is to be applied. We may, for example, give it a square-root program and the number 25, and expect it to deliver the number 5 to us. The computer "understands" the square-root program in the sense that it can interpret it in precisely the way we had in mind when we composed it. But then such a program converts a computer into a very special-purpose machine, a square-root-taking machine, and nothing more. Humans, if they are machines at all, are vastly general-purpose machines and, what is most important, they understand communications couched in natural languages (e.g., English) that lack, by very far, the precision and unambiguousness of ordinary programming languages. Since the over-all aim of AI is to build machines that are "responsive to the full detail of a library of encyclopedias," work must naturally be done to enable them to understand natural language. But, even apart from such dreams, there are both practical and scientific reasons for working on the natural-language problem. If people from outside the computer fields are to be able to interact significantly with computers, then either they must learn the computer's languages or it must learn theirs. Even now it is easier to give computers the jargon-laden languages of some specialists—e.g., some physicians, or researchers working on moon-rocks—than it is to train the specialists in the ordinary languages of computers. Some computer scientists believe their theories about language to be somehow not fully legitimate as long as they remain what the general public disdainfully calls "mere theories," that is, until it has been shown that they can be converted into models in computer-program form. On the other hand, many linguists, for example, Noam Chomsky, believe that enough thinking about language remains to be done to occupy them usefully for yet a little while, and that any effort to convert their present theories into computer models would, if attempted by the people best qualified, be a diversion from the main task. And they rightly see no point to spending any of their energies studying the work of the hackers.

But to the truly initiated member of the artificial intelligentsia, no reason for working on the problem of machine understanding of natural language need be stated explicitly. Man's capacity to manipulate symbols, his very ability to think, is inextricably interwoven with his linguistic abilities. Any re-creation of man in the form of machine must therefore capture this most essential of his identifying characteristics.

There is, of course, no *single* problem that can reasonably be called the natural-language problem for computers, just as there is

---

From *Computer Power and Human Reason,* Copyright © 1976 by W. H. Freeman and Co.

no such single problem for man. Instead, there are many problems, all having to do with enabling the computer to understand whatever messages are impressed on it from the world outside itself. The problem of computer vision, for example, is in many respects fundamentally the same as that of machine understanding of natural language. However the machine is made to derive information from its environment, it must, in some sense, "understand" it; that is, the computer must somehow be able to extract the semantic content from the messages that impinge on it, in part from their purely syntactic structure. It may seem odd, at first glance, to speak of the syntactic structure of a visual scene and to relate the process of understanding it to the process of understanding a natural-language text. But consider a picture of an adult and a child on a teetertotter. We understand certain aspects of that scene from its form, although even that understanding depends on our first having adopted a certain conceptual framework, a set of conventions. These conventions are syntactic in that they serve as criteria that permit us to distinguish legally admissable pictures, so to speak, from absurd ones. The ordinarily accepted picturing conventions would reject as ungrammatical most of the drawings of Escher, for example. We understand the teetertotter picture on the basis of semantic cues as well, however. We know, for example, that the adult figure, being down, is heavier than the child sitting high on the other side. And that knowledge comes to us from something other than the form of the picture, for it involves our private knowledge of aspects of the real world.

Language understanding, whether by man or machine, is like that too. We all have some criteria, an internalized grammar of the English language, that allow us to tell that the string of words "The house blue it" is ungrammatical. That is a purely syntactic judgement. On the other hand, we recognize that the sentence "The house blew it" is grammatical, even though we may have some difficulty deciding what it means, that is, how to understand it. We say we understand it only when we have been able to construct a story within which it makes sense, that is, when we can point to some contextual framework within which the sentence has a meaning, perhaps even an "obvious" meaning. For example, in a story in which a gambling house's scheme for beating a gambler's system misfired, the sentence "The house blew it" has a perfectly obvious meaning, at least to an American. Again, knowledge of the real world had to be brought to bear, not merely to disambiguate the sentence, but to assign meaning to it at all.

If we wish a machine to do something, we have to tell it to do it, and the machine must be able to understand what we say to it.

It is, of course, far easier to get a grip on the problem of machine understanding of natural language than on the corresponding problem for vision, first of all because language can be represented in written form, that is, as a string of symbols chosen from a very small alphabet. Moreover, such strings can be presented to the computer serially just as they are presented to human readers. They can also be stored with absolute fidelity. In contrast, the question of what constitutes a visual symbol, however primitive, already drags in major problems of both syntax and semantics. A worker on machine understanding of English text makes no important intellectual commitment to any particular research hypothesis or strategy when he adopts certain symbols as primitive, that is, as not themselves analyzable. But the worker on vision problems will have virtually determined major components of his research strategy the moment he decides on, say, edges and corners as elements of his primitive vocabulary. Besides, he faces a formidable problem just in deciding when his machine "sees" an edge. For this reason, as well as because it is only recently that television cameras have been coupled to computers, work on natural-language understanding by computers has a much longer history in artificial-intelligence research than does work on the problem of computer vision.

Either people must learn the computer's language or it must learn theirs.

It has happened many times in the history of modern computation that some technological advance in computer hardware or programming (software) has triggered a virtually euphoric mania. When what were then thought of as large-scale computers first began to work more or less reliably, some otherwise reasonable people fell victim to what I will call the "Now that we have *X* (at last), we can do *Y*" syndrome. In this situation the *X* was what were then considered very large information stores (memories) and very high computing speeds, and the *Y* was machine translation of languages. (I shall not cite references here, the fever that plagued the afflicted having long ago subsided.)

The early vision was, as Robert K. Lindsay was to later put it "that high quality translations could be produced by machines supplied with sufficiently detailed syntactic rules, a large dictionary, and sufficient speed to examine the context of ambiguous words for a few words in each direction." Computers are still not producing "high-quality translations." However, a hardcore dogmatist of the old school, if there is one left, might argue that we still don't have "sufficiently detailed syntactic rules" or "sufficient speed" to reach the desired end. But the real question is whether such sufficiency is possible at all. Would any set of syntactic rules, however detailed, and any computing speed, and any size dictionary suffice to produce high-quality translations? Every serious worker now agrees that the answer to this question is simply "no."

Translation must be seen as a process involving two distinct but not quite separable components: the text to be translated has to be understood; and the target-language text has to be produced. We can ignore the second of these components for our purposes here. The problem shows up in nearly its full complexity if we consider the target language to be the same as the source language and thus transform the translation problem into "simply" the paraphrasing problem. We have seen that to understand even a single sentence may involve both an elaborate contextual framework—e.g., a scenario having to do with gambling houses, gamblers' systems, and so on—and real-world knowledge—e.g., what gamblers do, what it means to break the bank, and so on. Let us return to that analogy. Suppose the sentence we cited, "The house blew it," occurred in the first chapter of a detective story. The detective's solution of the crime might hinge on his coming to understand that this sentence referred to gambling houses. But the clues that lead to that interpretation of the sentence may be revealed only gradually, say, one in each chapter. Then no man and no computer could be expected to understand, hence to paraphrase, that sentence when it first appears. Nor would an examination of a few words on either side of the sentence be any help whatever. Both a man and a computer would have to read all but the last chapter of the detective story to be able to do what the detective finally did, that is, understand the crucial sentence. (We assume that the story's last chapter serves only those who have missed a clue or have otherwise been unable to reach the appropriate conclusions.) And even then, only those with appropriate knowledge of the world could do it.

The recognition that a contextual framework is essential to understanding natural text was first exploited by so-called question-answering systems. B. F. Green and others wrote a system in 1961 that was able to understand and respond to questions about baseball, for example. It could understand the question "Where did each team play in July?" without difficulty because, in its universe of discourse, such possibly problematic words as "team" and "play" could have only unique meanings. It could answer because each unambiguously understood question could easily be converted into a small program for searching the system's data base for relevant information. Bobrow's program STUDENT, although very much more ambitious, exploited the same principle. It was able to solve so-called algebra word problems such as "Tom has twice as many fish as Mary has guppies. If Mary has three guppies, what is the number of fish Tom has?" Again, the universe of discourse within which the program was designed to operate determined how words and sentences were to be understood, reconstructed (into algebraic formulas), and otherwise manipulated. Note for example that, in order to "understand" the quoted problem, words like "fish" and "guppies" need not be "understood" at all; they could as well have been *"X"* and *"Y"*, respectively. And the word "has" has no connotation such as it would have in the sentence "Tom has a cold." The specification of a very highly constrained universe of discourse enormously simplifies the task of understanding—and that is, of course, true for human communication as well.

Obviously, understanding must be mutual in most realistic situations. In the context of man-machine communication, we wish the machine to understand us in order that it may do something for us, e.g., answer a question, solve a mathematical problem, or navigate a vehicle, which action we, in turn, hope to understand. The examples just cited shed no light on this aspect of man-machine communication. The answers delivered by either the BASEBALL or the STUDENT program simply do not have sufficient interpretive scope to be problematical. One cannot imagine having an interesting conversation with them. Among other things, and most significantly, they do not themselves ask questions.

The first program that illuminated this other side of the man-machine communication problem was my own ELIZA. (I chose the name "Eliza" because, like G. B. Shaw's Eliza Doolittle of *Pygmalion* fame, the program could be taught to "speak" increasingly well, although, also like Miss Doolittle, it was never quite clear whether or not it became smarter.) ELIZA was a program consisting mainly of general methods for analyzing sentences and sentence fragments, locating so-called key words in texts, assembling sentences from fragments, and so on. It had, in other words, no built-in contextual framework or universe of discourse. This was supplied to it by a "script." In a sense ELIZA was an actress who commanded a set of techniques but who had nothing of her own to say. The script, in turn, was a set of rules which permitted the actor to improvise on whatever resources it provided.

The first extensive script I prepared for ELIZA was one that enabled it to parody the responses of a nondirective psychotherapist in an initial psychiatric interview. I chose this script because it enabled me to temporarily sidestep the problem of giving the program a data base of real-world knowledge. After all, I reasoned, a psychiatrist can reflect the patient's remark, "My mommy took my teddy bear away from me," by saying, "Tell me more about your parents," without really having to know anything about teddy bears, for example. In order to generate this response, the program had to know that "mommy" means "mother" and that the "patient" was telling it something about one of his parents. Indeed, it gleaned more than that from the subject's input, some of which, for example, it might use in later responses. Still, it could have been said to have "understood" anything in only the weakest possible sense.

Any re-creation of man in the form of machine must capture man's ability to think.

Nevertheless, ELIZA created the most remarkable illusion of having understood in the minds of the many people who conversed with it. People who knew very well that they were conversing with a machine soon forgot that fact, just as theatergoers, in the grip of suspended disbelief, soon forget that the action they are witnessing is not "real." This illusion was especially strong and most tenaciously clung to among people who knew little or nothing about computers. They would often demand to be permitted to converse with the system in private, and would, after conversing with it for a time, insist, in spite of my explanations, that the machine really understood them.

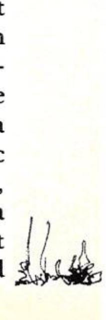
This phenomenon is comparable to the conviction many people have that fortune-tellers really do have some deep insight, that they do "know things," and so on. This belief is not a conclusion reached after a careful weighing of evidence. It is rather a hypothesis which, in the minds of those who hold it, is confirmed by the fortune-teller's pronouncements. As such, it serves the function of the drunkard's lamppost we discussed earlier: no light is permitted to be shed on any evidence that might be disconfirming; and, indeed, anything that might be seen as such evidence by a disinterested observer is interpreted in a way that elaborates and fortifies the hypothesis.

Suppose the statement "The house blew it" occurred in the first chapter of a detective story. . . .

Within limits, this is a quite normal and even necessary process. No "information" is data except in the light of some hypothesis. Therefore, even in an ordinary two-person conversation, each participant brings something of himself to bear on the process of understanding the other. Each has, in other words, a working hypothesis, again a conceptual framework, concerning who the other is and what the conversation is about. This hypothesis serves as a predictor of what the other is going to say and, more importantly, of what he intends to mean by what he is going to say. This predictor functions simultaneously at several distinct levels. On the lowest level, a listener anticipates what the speaker's next few words will be; he completes yet unfinished sentences for him. If, for example, one is on an elevator and hears the operator say "This elevator does not stop on the . . . ," one would expect him to complete the sentence with the word "floor," but not with "second Thursday of each month." Sometimes the listener predicts wrongly and repairs the resulting damage only much later in the conversation, that is, when overwhelming evidence that he must have "misheard" is presented to him. Often, however, the erroneous prediction is falsified before the sentence in question has been completed by the speaker. The listener then makes corrections on the fly and virtually unconsciously.

On a much higher level, each participant brings to the conversation an image of who the other is. Since it is impossible for any human to know another completely, that image consists in part of attributions to the other's identity, attributions which must necessarily be based on evidence derived from independent life experiences of the participant. Our recognition of another person is thus an act of induction on evidence presented to us partly by him and partly by our reconstruction of the rest of the world; it is a kind of generalization. We are, in other words, all of us prejudiced—in the sense of pre-judging—about each other. And, as we have noted, we all find it hard, or even nearly impossible, to perceive—let alone to accept and to permit to become operative—evidence that tends to disconfirm our judgments.

It is then easy to understand why people conversing with ELIZA believe, and cling to the belief, that they are being understood. The "sense" and the continuity the person conversing with ELIZA perceives is supplied largely by the person himself. He assigns meanings and interpretations to what ELIZA "says" that confirm his initial hypothesis that the system does understand, just as he might do with what a fortune-teller says to him. All ELIZA or the fortune-teller need do is give responses that are sufficiently plausible and that allow a sufficient scope for interpretation to make such constructions possible. And, since the subject cannot probe the true limits of ELIZA's capacities (he has, after all, only a limited time to play with it, and it is constantly getting new material from him), he cannot help but attribute more power to it than it actually has. Besides, he knows that ELIZA was constructed by a professor at a university. It is therefore clothed in the magical mantle of Science and all of Science's well-known powers may be attributed to it.



ELIZA did, in fact, generate plausible responses to what was said to it. In order to be able to do that, it too had to be supplied with a set of expectations. These were encoded in whatever script was given to it. A person playing with ELIZA in its psychiatrist mode was instructed to provide ELIZA with the sort of statements one might make to a psychiatrist in an initial psychiatric interview. He was told, in other words, what ELIZA's expectations were. On a lower level, ELIZA's psychiatric script was constructed in a way that allowed

ELIZA to make local predictions about sentences and textual fragments, that is, to apply hypotheses to them which further examinations might confirm or falsify. For example, the psychiatric script entertained the initial hypothesis that a fragment of the general form "everybody...me," although patently conveying a message about the subject's relationship to "everybody," e.g., "everybody hates me," or about what everybody is doing to the subject, e.g., "everybody is always laughing at me," latently and more importantly referred to a recent incident involving the subject and only a single or at most a few individuals. ELIZA's response might therefore be "Tell me, who told you he hated you within the last few days?" or "Who laughed at you recently?"

What sharply distinguishes the current work on machine understanding of natural language from the work of the early 1960's and before is precisely the current strong use of prediction, both on the local syntactic level and, more importantly, on the larger contextual level. Roger C. Schank, an exceptionally brilliant young representative of the modern school, bases his theory on the central idea that every natural-language utterance is a manifestation, an encoding, of an underlying conceptual structure. Understanding an utterance means encoding it (Schank uses the technical term "mapping") into one's own internal conceptual structure.

Many times in the history of modern computation some technological advance has triggered a virtually euphoric mania.

> Any two utterances that can be said to mean the same thing, whether they are in the same or different languages, should be characterized in only one way by the conceptual structures.... The representation of this conceptual content then, must be in terms that are interlingual and as neutral as possible.... We will be... [concerned] with finding, once something is said, a representation that will account for the meaning of that utterance in an unambiguous way and one that can be transformed back into that utterance or back into any other utterances that have the same meaning.
>
> The important point is that underlying every sentence in a language, there exists at least one conceptualization.

What I wish to emphasize here is that Schank's theory proposes a formal structure for the conceptual bases underlying linguistic utterances, that it proposes specific mechanisms (algorithms) for basing predictions on such conceptual structures, and that it proposes formal rules for analyzing natural-language utterances and for converting them into the conceptual bases. However, Schank does not believe that an individual's entire base of conceptions can be explicitly extricated from him. He believes only that there exists such a belief structure within each of us, and that, if it could be explicated, it could in principle be represented by his formalism. One difficulty, which Schank of course recognizes, is that every individual's belief structure is constantly changing.

In discussing the role a person's belief structure plays in the way he participates in conversations, I wrote in my 1967 "Contextual Understanding" paper:

> In some areas of the individual's intellectual life, this structure may be highly logically organized—at least up to a point; for example, in the area of his own profession. In more emotionally loaded areas, the structure may be very loosely organized and even contain many contradictions. When a person enters a conversation he brings his belief structures with him as a kind of agenda.
>
> A person's belief structure is a product of his entire life experience. All people have some common formative experiences, e.g., they were all born of mothers. There is consequently some basis of understanding between any

two humans simply because they are human. But even humans living in the same culture will have difficulty in understanding one another where their respective lives differed radically. Since, in the last analysis, each of our lives is unique, there is a limit to what we can bring another person to understand. There is an ultimate privacy about each of us that absolutely precludes full communication of any of our ideas to the universe outside ourselves and which thus isolates each one of us from every other noetic object in the world.

There can be no total understanding and no absolutely reliable test of understanding.

To know with certainty that a person understood what has been said to him is to perceive his entire belief structure and *that* is equivalent to sharing his entire life experience. It is precisely barriers of this kind that artists, especially poets, struggle against.

This issue must be confronted if there is to be any agreement as to what machine "understanding" might mean. What the above argument is intended to make clear is that it is too much to insist that a machine understands a sentence (or a symphony or a poem) only if that sentence invokes the same imagery in the machine as was present in the speaker of the sentence at the time he uttered it. For by that criterion no human understands any other human. Yet, we agree that humans do understand one another to *within acceptable tolerances.* The operative word is "acceptable" for it implies *purpose.* When, therefore, we speak of a machine understanding, we must mean understanding as limited by some objective. He who asserts that there are certain ideas no machines will ever understand can mean at most that the machine will not understand these ideas tolerably well because they relate to objectives that are, in his judgment, inappropriate with respect to machines. Of course, the machine can still deal with such ideas symbolically, i.e., in ways which are reflections—however pale—of the ways organisms for which such objectives are appropriate deal with them.

One cannot imagine having an interesting conversation with machines because they do not themselves ask questions.

I would expect Schank, as well as most other workers now in this field, to find this consistent with their own ideas. However, when I used the term "imagery" ("that sentence invokes the same imagery"), a term I now see as roughly corresponding to Schank's "conceptual structures," I had no idea at all about how such images might be represented in a formal system.

There exists today several computer language-understanding systems that rely on ideas that resemble Schank's much more than superficially, even though they were arrived at independently and do differ in important respects. One of the best of these, and also one of the better known, is that of Terry Winograd. Winograd, at the time a graduate student in M.I.T.'s Artificial Intelligence Laboratory, was working in close association with a group building a computer-controlled "hand-eye" machine, that is, a computer system that could "see" its environment by means of an attached television camera, and that could manipulate objects within its environment by means of a computer-controlled mechanical arm and hand. One problem this system solved—to give an example of its general capability—was to copy a structure made of children's blocks, e.g., one consisting of arches, bridges, and so on, on the basis of a model of the desired structure that had been shown to it. Winograd addressed himself to the task of developing a system to enable humans to instruct this robot, strictly with respect to the toy world of blocks within which it was to operate, in natural language.

> Within this mini-world, we can give the computer a deep kind of knowledge.... [The robot] can manipulate toy blocks on a table containing simple objects like a box. In the course of a dialogue, it can be asked to manipulate the objects, doing such things as building stacks and putting things into the box. It can be questioned about the current configurations of blocks on the table, about the events that have gone on during the discussion, and to a limited extent about its reasoning. It can be told simple facts which are added to its store of knowledge for use in later reasoning. The conversation goes on within a dynamic framework—one in which the computer is an active participant, doing things to change his toy world, and discussing them.

Following is a sample conversation with the robot.

Human: Find a block which is taller than the one you are holding and put it into the box.

Robot: By "it," I assume you mean the block which is taller than the one I am holding.

H: OK.

R: (The hand sets down the block it is holding and picks up a taller one.)

H: What does the box contain?

R: The blue pyramid and the blue box.

H: What is the pyramid supported by?

R: The box.

...

H: Why did you clear off [the red] cube?

R: To put it on a large green cube.

H: Why did you do that?

R: To stack up the red cube, a large red block, and a large green cube.

H: Why did you do that?

R: Because you asked me to.

Winograd's program is an important achievement. It shows how a specific view of certain aspects of language can really be filled in with enough detail to provide a working model. It consists fundamentally of procedural descriptions, i.e., programs, of cognitive processes and is thus an embodiment of a particular psychological approach to language understanding as a process. On the other hand, the program has serious shortcomings, some of which Winograd himself notes:

> [It does not deal] with all the implications of viewing language as a process of communication between two intelligent people. A human language user is always engaged in a process of trying to understand the world around him, including the person he is talking to. He is actively constructing models and hypotheses, and he makes use of them in the process of language understanding. . . .
>
> [Because the system] keeps track of when things have been mentioned, it can check a possible interpretation of a question to see whether the asker could answer it himself from his previous sentences. If so, it assumes he probably means something else. We could characterize this as containing two sorts of knowledge. First, it assumes that a person asks questions for the purpose of getting information he doesn't already have, and second, it has a very primitive model of what information he has on the basis of what he has said. A realistic view of language must have a complex model of this type, and the heuristics in our system touch only the tiniest bit of the relevant knowledge.

ELIZA created the most remarkable illusion of having understood.

It really must be said that this expression of humility is enormously refreshing, especially since it comes from within the priesthood of the artificial intelligentsia, and so is virtually unique. Unfortunately, it does not go far enough. For what Winograd has done—indeed, what all of artificial intelligence has so far done—is to build a machine that performs certain specific tasks, just as, say, seventeenth-century artisans built machines that kept time, fired iron balls over considerable distances, and so forth. Those artisans would have been grievously mistaken had they let their successes lead them to the conclusion that they had begun to approach a general theoretical understanding of the universe, or even to the conclusion that, because their machines worked, they had validated the idea that the laws of the universe are formalizable in mathematical terms. The *hubris* of the artificial intelligentsia is manifested precisely by its constant advance of exactly these mistaken ideas about the machines it has succeeded in building. Neither Winograd's humility, nor that of any other AI researcher, extends to that admission.

Newell, Simon, Schank, and Winograd simply mistake the nature of the problems they believe themselves to be "solving." As if they were benighted artisans of the seventeenth century, they present "general theories" that are really only virtually empty heuristic slogans, and then claim to have verified these "theories" by constructing models that do perform some tasks, but in a way that fails to give insight into general principles. The failure is intrinsic, for they have failed to recognize that, in order to do what they claim to do, they must discover and formulate general principles of more power than that inherent in the observation, or even the demonstration, that laws can be stated in the form of computer programs. The most important and far-reaching effect of this failure is that researchers in artificial intelligence constantly delude themselves into believing that the reason any particular system has not come close to realizing AI's grand vision is always to be found in the limitations of the specific system's program. Thus, for example, Winograd acknowledges that his system avails itself of only the tiniest bit of relevant knowledge. The knowledge he is talking about is the knowledge of "facts" that is available to humans. But the problem with his approach is that his heuristics express no interesting general principles. Furthermore, such principles cannot be discovered merely by expanding the range of a system in a way that enables it to get more knowledge of the world. Even the most clever clock builder of the seventeenth century would never have discovered Newton's laws simply by building ever fancier and more intricate clocks!

Artificial intelligence has, as we have documented, set as its goal the building of machines whose range of thought is to be coextensive with that of humanity itself. (Never mind, for now, whether this is to be achieved in the "visible future" or not.) And the theories that are to underpin this triumph of AI are to apply to the whole man as well. Clearly, then, the kinds of limitations to which presently existing systems are subject, and to some of which Winograd confesses with genuine humility, are seen by the AI community as a whole as being merely temporary difficulties that can be overcome—in the visible future, according to Newell and Simon. There are then, two questions that must ultimately be confronted. First, are the conceptual bases that underlie linguistic understanding entirely formalizable, even in principle, as Schank suggests and as most workers in AI believe? Second, are there ideas that, as I suggested, "no machines will ever understand because they relate to objectives that are inappropriate for machines"?

When a person enters a conversation, he brings his belief structures with him as a kind of agenda.

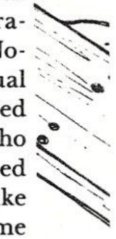
These two questions are of enormous importance. They go to the heart of

the question about whether there is any essential difference between man and machine. And it is appropriate that they be asked in the context of a discussion of the problem of natural-language understanding by machines, for it is in his language, above all, that man manifests his intelligence and, some believe, his unique identity as man. The two questions also need to be asked together. They are inextricably linked to one another. For if the whole of a human experience and the belief structure to which it gives rise cannot be formalized, then there are indeed appropriately human objectives that are inappropriate for machines. And if we were to conclude (as I intend to) that there are indeed such objectives, then we could also say something about what machines ought and ought not to be put to doing.

The fact that these questions have become important at all is indicative of the depth to which the information-processing metaphor has penetrated both the academic and the popular mind. For when we take stock, we quickly discover how little has actually been accomplished so far. Newell and Simon's book, *Human Problem Solving,* speaks in detail of only three problems: cryptarithmetic, theorem proving in the simplest logical calculus, and chess. The accomplishments of the computer-controlled "hand-eye" machines at MIT and Stanford University (e.g., building block structures from models shown to them, and screwing nuts onto bolts), are rightly hailed as triumphs by those who understand the incredible complexity of the problems that had first to be solved. And there have been other triumphs of similar magnitude. But the very fact that such achievements deserve to be so applauded itself testifies to how utterly primitive is our current knowledge about the human mind. George A. Miller, like the modern computer linguists, also speaks of the conceptual structures that underlie human thought and language, but he says, "To pretend that we know how to impart these complex conceptual structures to any machine at the present time is simply absurd." Winograd is really speaking for the entire field of artificial intelligence, whether his colleagues acknowledge that fact or not, when he says that our systems have touched only the tiniest bit of the relevant knowledge.

Is the computer capable of being desperate?

But just as it would have been unfair to argue in the seventeenth century that to place a manmade object into Earth orbit is impossible on the grounds that no one at that time had the slightest idea about how to accomplish it, so it would be wrong today to make impossibility arguments about what computers can do entirely on the grounds of our present ignorance. It is relevant, however, especially for taking stock of our present situation, to examine the power of the theories we have been discussing. Are they, for example, Newtonian in the vastness of their inferential scope?



What is contributed when it is asserted that "there exists a conceptual base that is interlingual, onto which linguistic structures in a given language map during the understanding process and out of which such structures are created during generation [of linguistic utterances]"? Nothing at all. For the term "conceptual base" could perfectly well be replaced by the word "something." And who could argue with that so-transformed statement? Schank's contribution, like those of others now tilling the same fertile fields, is that he attempts to provide a formal representation of the conceptual base. He intends to tell us in utmost detail what that something "that underlies all natural languages" is and how it functions in both the generation and the understanding of linguistic utterances. Even then, Schank provides no demonstration that his scheme is more than a collection of heuristics that happen to work on specific classes of examples. The crucial scientific problem would be to construct a finite program that assigns appropriate conceptual structures to the infinite range of sentences that can occur in natural language. That problem remains as untouched as ever. Imagine an adding machine that adds some but not all numbers correctly, and about which we can't even say what characterizes the numbers it can add. We would hardly call that a

mechanization of arithmetic. And in what form are the conceptual structures Schank hypothesizes, and the operations on them, linkages among them, and so on, to be shown to us? In the form of computer programs, of course.

> What is needed, and what has been lacking, is a cohesive theory of how humans understand natural language without regard to particular subparts of that problem, but with regard to that problem as a whole. The theory . . . is also intended to be a basis for computer programs that understand natural language. . . . What will be discussed is the theory of such a program. . . .
>
> We hope to be able to build a program that can learn, as a child does, how to do what we have described in this paper instead of being spoon-fed the tremendous information necessary. In order to do this it might be necessary to await an effective automatic hand-eye system and an image processor.

The theories hypnotizing the artificial intelligentsia determine that life is what is computable and only that.

Here we begin to see the confluence of the work on problem solving we discussed earlier and the work on natural-language understanding by machine. A sentence is a "given object," the conceptual structure that is its meaning is the "desired object," the goal is to transform the former into the latter, and the means are those provided by the understanding program. But what is most important in both instances is that the theories be convertible to computer programs.

It may be possible, following Schank's procedures, to construct a conceptual structure that corresponds to the meaning of the sentence, "Will you come to dinner with me this evening?" But it is hard to see—and I know that this is not an impossibility argument—how Schank-like schemes could possibly understand that same sentence to mean a shy young man's desperate longing for love. Even if a computer could simulate feelings of desperation and of love, is the computer capable of *being* desperate and of loving? Can the computer then understand desperation and love? To the extent that those are legitimate questions at all, and that is a very limited extent indeed, the answer is "no." And if that is the answer, then the sense in which even the most powerful Schank-like system "understands" is about as weak as the sense in which ELIZA "understood."

At best, what we see here is another example of the drunkard's search. A theory purports to describe the conceptual structures that underlie all human language understanding. But the only conceptual structures it admits as legitimate are those that can be represented in the form of computer-manipulatable data structures. These are then simply pronounced to constitute all the conceptual structures that underlie all of human thought. Given such a program, i.e., such a narrowing of the meaning of the word "all," it should indeed be possible to prove that the theory accounts for "all" human linguistic behavior!

A theory is, of course, itself a conceptual framework. And so it determines what is and what is not to count as fact. The theories—or, perhaps better said, the root metaphors—that have hypnotized the artificial intelligentsia, and large segments of the general public as well, have long ago determined that life is what is computable and only that. As Professor John McCarthy, head of Stanford University's Artificial Intelligence Laboratory said, "The only reason we have not yet succeeded in simulating every aspect of the real world is that we have been lacking a sufficiently powerful logical calculus. I am currently working on that problem."

Sometimes when my children were still little, my wife and I would stand over them as they lay sleeping in their beds. We spoke to each other in silence, rehearsing a scene as old as mankind itself. It is as Ionesco told his journal: "Not everything is unsayable in words, only the living truth." ▼

Microcomputers Help the Deaf-Blind

by Frederick M. Kruger, Ph.D.
Director of Research,
Helen Keller National Center for
Deaf-Blind Youths and Adults

We are in a period of rapid technological advancement which has already shown us that what truly seemed to be impossible just a few short years ago is now taken for granted. Only twenty years ago a person needed to pay close to a hundred dollars for a Zenith transistor radio. Today one can purchase a much better and more efficient receiver for only five to ten percent of that figure. Twenty years ago one spoke of IBM and UNIVAC computers and the multimillion-dollar companies and government agencies which leased them. Today we speak of purchasing large-scale integrated microcomputer chips with far greater utilization potential for significantly less than the cost of that 1957-vintage Zenith transistor radio. Complete, ready-to-operate, sophisticated interactive home computer systems with various forms of bulk storage are available for less than two thousand dollars. Two manufacturers have even produced ready-to-run home computers for less than six hundred dollars, selling them along with comparably priced high-fidelity systems, calculators, and citizens' band radio equipment. Why is it, then, that we have been so slow in applying these phenomenal developments to the needs of many severely handicapped persons?

Deaf-blindness is a fact of life for ten to fifteen thousand people in the United States. With no means of distinguishing between day and night, no radio or TV, and minimal access to magazines or newspapers, most of these people live a very lonely, isolated life. Because of the relatively small number of such persons, and the circumstances surrounding their lives, deaf-blind individuals belong to an all but invisible minority. Even though some of these people have some sight or hearing, their use of it is, at best, quite limited. This article will give emphasis to the problems of the deaf-blind person with no usable vision or hearing.

Deaf-blind people are like everyone else in most respects. They too have needs and desires, and they too like to communicate. Intellect is not a func-

Photographs courtesy of the author

One Hand Manual Alphabet (fingerspelling)

tion of visual or auditory ability. Most deaf-blind people can learn to communicate, some quite well, using other channels.

Persons who are deaf and persons who are deaf-blind have many communication problems in common. When they are close together, persons with either handicap communicate through various combinations of touch and gesture. The deaf person may have someone attract his attention through a touch on the shoulder. He would then look at the other person and carry on a conversation through sign language or fingerspelling (One Hand Manual Alphabet) or even lipreading. The two might even write messages to each other.

In contrast, the deaf-blind person could only receive the signs or spelled messages tactually, that is, by holding onto the other person's hands and following the signs or gestures, or by having words fingerspelled into his hand. Writing messages back and forth is very difficult, if not impossible. Sometimes a gentle push or other contact gesture is sufficient for communication. For example, a deaf-blind person might learn that a large X written across his back is an emergency signal meaning "go to the emergency exit immediately." Although this last technique is used, it leaves much to be desired.

Think for a moment of the problem that would exist if you wished to tell the deaf-blind person that there was a fire and he should go to the exit, but the fire separated you from the deaf-blind person—and each had a clear path to different assigned exits. If he could see, a signal might not be needed, or brightly flashing alarm lights might accomplish the desired goal. If he could hear, alarm bells would be sufficient. Unfortunately, without vision or hearing, the deaf-blind person would be totally alone—even in a crowded room—if no one were near enough to touch him!

Beyond the span of three feet, a distance approximating the span of two outstretched arms, the problems of communication for the deaf-blind individual become immense. Every communication activity is, for the deaf-blind person, a specialized activity requiring special equipment. When personal contact is not possible, communication can frequently require very advanced technology!

What *are* the special requirements to be considered when defining the beyond-three-feet communication needs of the deaf-blind person? What are the various communication devices a deaf-blind person might need or want to use? What devices are necessary so as not to deprive deaf-blind persons of the everyday conveniences we inevitably take for granted?

Beyond the span of two outstretched arms, the problems of communication for the deaf-blind individual become immense.

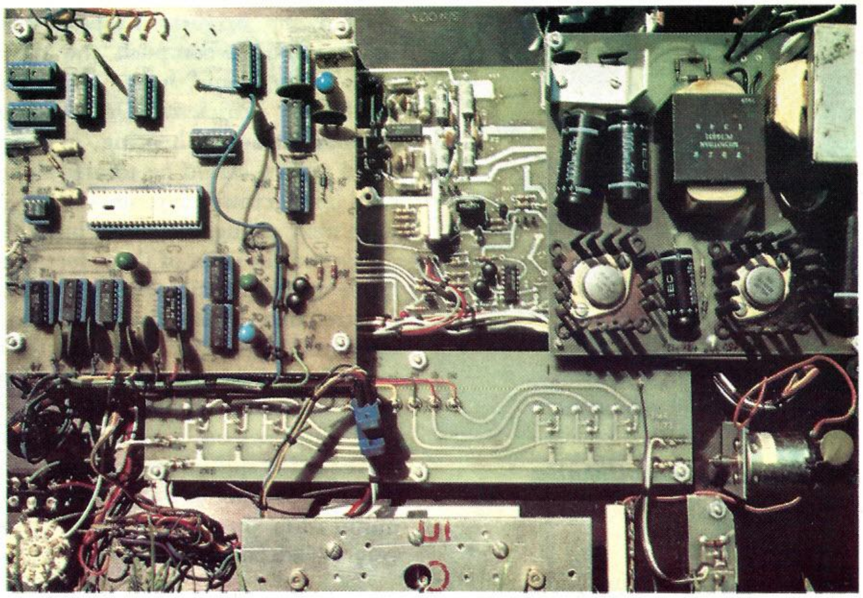
The first of these requirements is the need for special *input-output* conversion. What specific provisions must be made to convert a standard device output (for example, sound from a telephone) so that it can be adequately perceived by the deaf-blind person? Also, what specific provisions must be made to convert a potentially nonstandard output from the deaf-blind person to the standard input that may be required by a device (for example, sound to vibration, button presses to sound, Braille to ASCII, Baudot to Braille, ASCII to Braille, or Braille to Baudot)?

The second requirement is ease of operation. Any special communication device must be designed for use by average people—not geniuses! Moreover, under emergency conditions even a genius may become confused and be unable to properly operate an overly complex unit. With ease of operation comes the requirement that the device not intrude on other normal activities. It should not require special handling or attention when it is not in use. Also, most deaf-blind people would prefer that all such devices be small and sufficiently inconspicuous not to call attention to them—especially when not in use. Obviously, this isn't always possible.

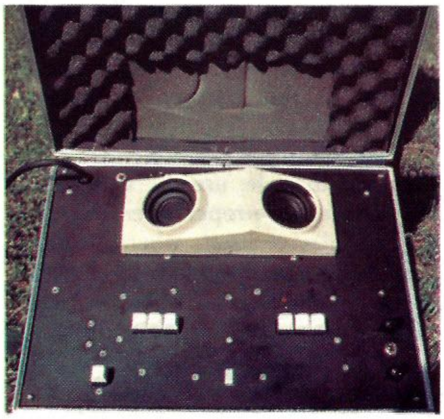
The third requirement is reliability. *Any* aid or device designed for deaf-blind people, no matter what its specific application, must be reliable. It must always operate when called upon to do its job; it must be sufficiently rugged to survive falls from shelves or table tops; and, where at all possible, it should be very water resistant. Also, if it uses a battery, it should not be ruined if the battery is inserted backwards.

Two additional desirable considerations are that the user be able to test the device for proper operation and that the device itself, where possible, monitor its own operation and indicate in some unique manner when future reliable operation is questionable. For example, a child's hearing aid might be considered more valuable if it began to flash a small indicator light a few hours before its batteries died. This would indicate to a parent or instructor that the battery needed replacement

A deaf-blind person is reading from a magazine. Notice the watch on her left hand. It has a pop-up lid, sturdy hands, and raised dots at principle points so that it can be read by touch.



Note the motor at the lower right in this bottom view of the field-test-model Telebraille. It has an off-balance weight on its shaft and vibrates when it is activated.

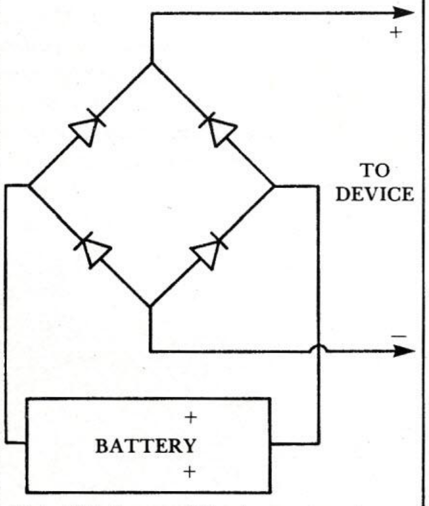


In this front view of the Telebraille, the acoustic coupler is at the top center, the six Braille keys along the midline, and the Braille cell at the bottom center. The interrupt button is at the bottom left; the controls for originate/answer mode, Braille display duration, and self-test/operate are at the right side. The Telebraille is now being field-tested by a number of deaf-blind persons. On 14 October 1977 two Telebrailles—one at the Helen Keller National Center in Sands Point, Long Island, and the other in Chicago, Illinois—were acoustically coupled to standard telephones and Braille was transmitted in both directions—over a thousand-mile path!

Diodes may be used to electronically switch battery polarity.

Device will receive correct polarity voltage no matter how the battery is inserted. (Better yet, make it impossible to insert the battery backwards!)

If this circuit is used, the 1.2 volt drop through the two silicon diodes must be considered when specifying the required battery voltage.



Note: This is a good idea for any experimental device where there is a risk of hooking up a bench power supply backwards while rushing or when tired!

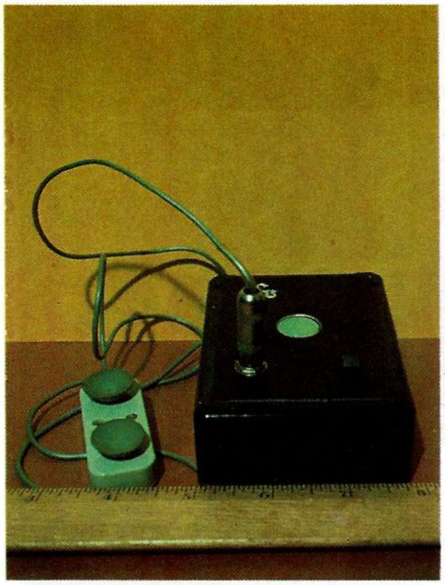
and would avoid the problem of a child wearing a "dead" aid for some period of time.

In similar fashion, any electronic communication aid for a deaf-blind person should indicate imminent failure. Since the user may not be able to see or hear at all, a unique vibratory signal might be triggered to indicate a low-power condition, etc.

Until recently, the use of a telephone by a deaf-blind person required quite a bit of ingenuity. If he could speak intelligibly, for example, he could call a person, explain that he could not hear, and instruct the distant individual to respond to various questions by operating the rotary telephone dial to make one click for "no" and four clicks for "yes." (Obviously, this technique could not be used with Touch-Tone telephones.) The hearing person might instead be instructed to answer questions with "no" or "yes-yes." The caller would place his finger on the earpiece, or on the output transducer of a device designed to amplify these clicks, so that he could feel the coded replies to his questions. (A hearing aid with a telephone pickup coil and a bone conduction transducer could be used.) Thus, through a series of questions similar to the game of Twenty Questions, the deaf-blind person could converse. If the deaf-blind person and the distant individual both knew Morse code, they could converse more normally. The distant person would either dial a 1 for a dot and a 4 for a dash (one click and four clicks) or say "dit" for a dot and "da-a" for a dash. The different duration vibrations would be felt on the output transducer.

A child's hearing aid might be more valuable if it flashed a small light a few hours before its batteries died.

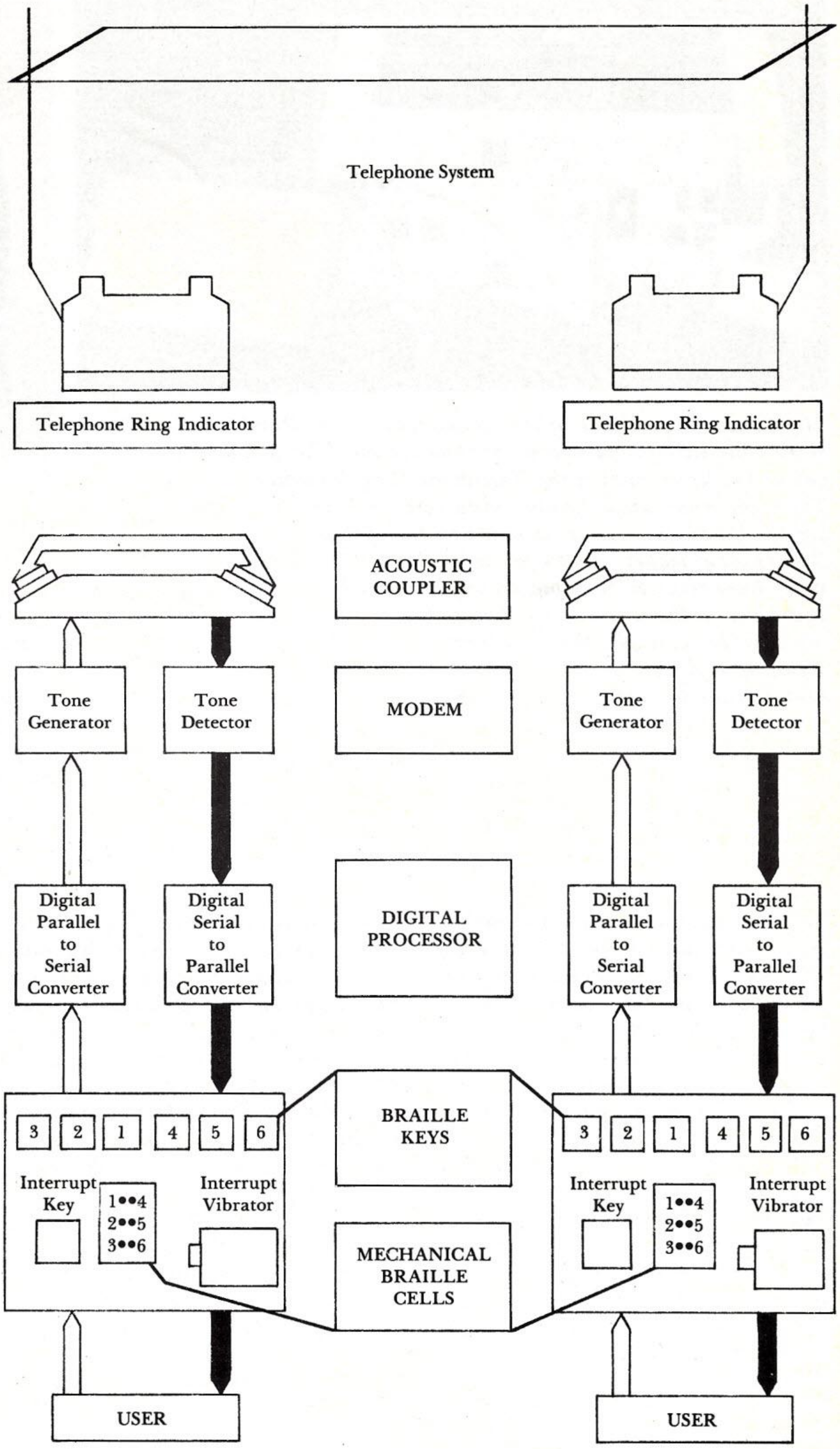
For the person who can't feel the vibrations directly from the telephone earpiece (receiver) and who doesn't



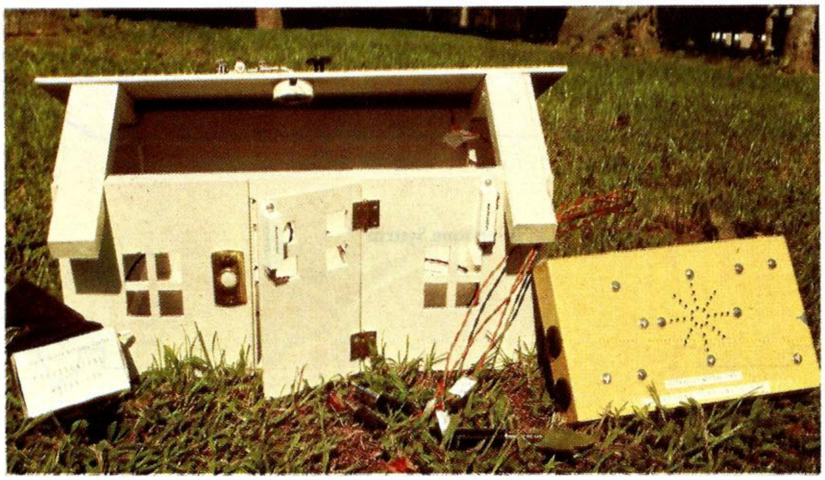
The Tactile Speech Indicator model shown here was manufactured by the American Foundation for the Blind. The induction coil, to the left, is attached to the telephone earpiece and feeds signals to the Tactile Speech Indicator.

own a hearing aid with the telephone coil and bone vibrator, a special portable electronic device was designed. It is called the Tactile Speech Indicator, and it works quite like a hearing aid with an "induction pickup" telephone coil and a bone vibrator. A similar device, which is directly attached to the telephone, provides for sending and receiving Morse code more easily. It is called the Code-Com and is available through many Bell Telephone Company business offices.

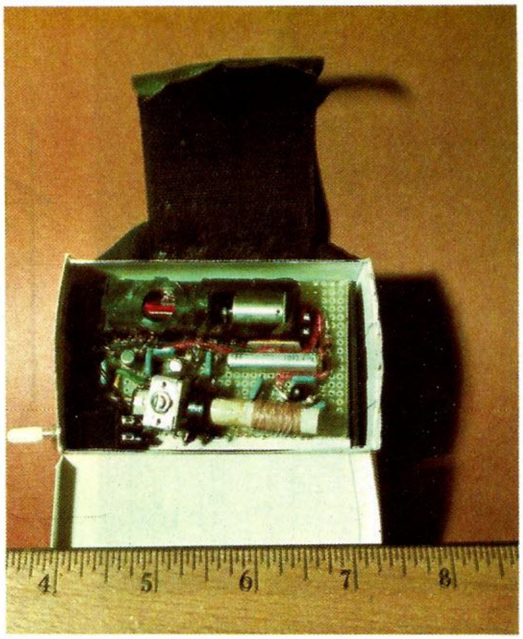
With these beginnings, the use of the telephone by many deaf-blind persons as a communication-at-a-distance device became a reality. But what about the deaf-blind person who could not speak intelligibly or did not know Morse code? What could he do? Unfortunately, the most frequent answer was "Not too much!" The fact is that it is rarely, if ever, profitable for a manufacturer to make devices for so small a group of potential users. In spite of partial or sometimes even total government support for the development of special communication devices, the cost to manufacture and sell them commercially—including parts, labor, overhead, profits, etc.—usually puts


Telephone System
Telephone Ring Indicator
Telephone Ring Indicator
ACOUSTIC COUPLER
Tone Generator
Tone Detector
MODEM
Tone Generator
Tone Detector
Digital Parallel to Serial Converter
Digital Serial to Parallel Converter
DIGITAL PROCESSOR
Digital Parallel to Serial Converter
Digital Serial to Parallel Converter
3 2 1 4 5 6
BRAILLE KEYS
3 2 1 4 5 6
Interrupt Key
1••4
2••5
3••6
Interrupt Vibrator
MECHANICAL BRAILLE CELLS
Interrupt Key
1••4
2••5
3••6
Interrupt Vibrator
USER
USER

BLOCK DIAGRAM OF THE TELEBRAILLE AS IT PRESENTLY EXISTS



This picture shows some of the components of the Residential Wrist-Com wireless vibratory signaling system. The yellow box at the lower right is the Telephone Ring Indicator. There are transparent Braille characters over the printed labels. The device is made with a special DYMO label maker. At the lower left is the Wrist-Com receiver (completely self contained and worn on the wrist). The "house" contains the doorbell, burglar alarm sensor, and fire sensor for demonstration purposes.



This is a view of the hand-wired prototype Residential Wrist-Com receiver. Toward the top is the battery holder/motor-vibrator unit. Just below the motor is a narrow band-pass tone filter. (Both of these components are manufactured by Motorola.) The antenna coil can be seen below the filter. During recent tests, one N size, 1.5-volt mercury battery lasted for more than two hundred hours of continuous use—silent periods and *lots of demonstrations.*

the purchase price beyond the reach of most deaf-blind people. This discourages technological innovation and many good ideas are never developed because of the lack of adequate support. We hope that new technologies and innovative manufacturing techniques will alter this picture soon.

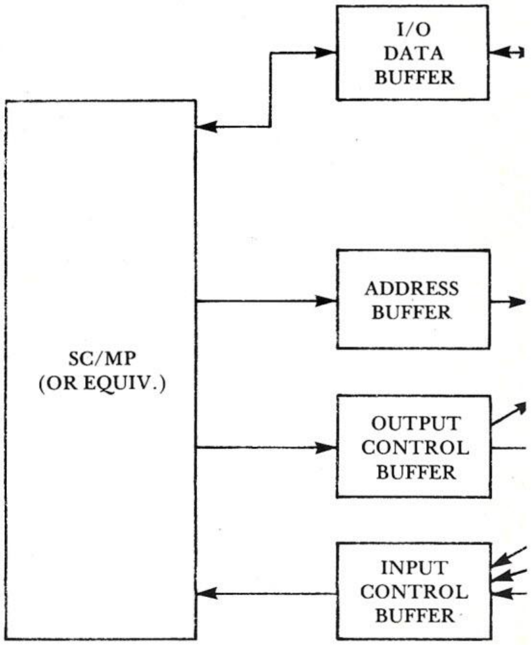
Several projects have been initiated at the Helen Keller National Center for Deaf-Blind Youths and Adults which have as primary goals the development of good communication aids for survival purposes. Obviously, it is our implicit intention to provide the means for better social communication as well—even if still only on a one-to-one basis.

The following is a somewhat detailed discussion of the devices we are presently developing, followed by an outline of some of our plans for the future. The technical designs for and information about any devices we have developed or are developing will be made available to any responsible individual or group wishing to produce them for deaf-blind persons. We only insist that our quality standards be maintained and that the selling price to the ultimate consumer be as low as reasonably possible and consistent with our guidelines. Consulting services are also provided without charge when requested.

Presently, three devices are being developed at the Helen Keller Center. They are the Telebraille, the residential model of the Wrist-Com, and the institutional model of the Wrist-Com.

Telebraille

The Telebraille was designed to enable two deaf-blind people to communicate over the telephone using Braille. There is no need for third-party intervention. The "speaker" sends Braille using a keyboard quite similar to that on a standard Brailling machine (a



standard typewriter keyboard will also be available in the future). The "listener" reads the message from a single Braille cell.

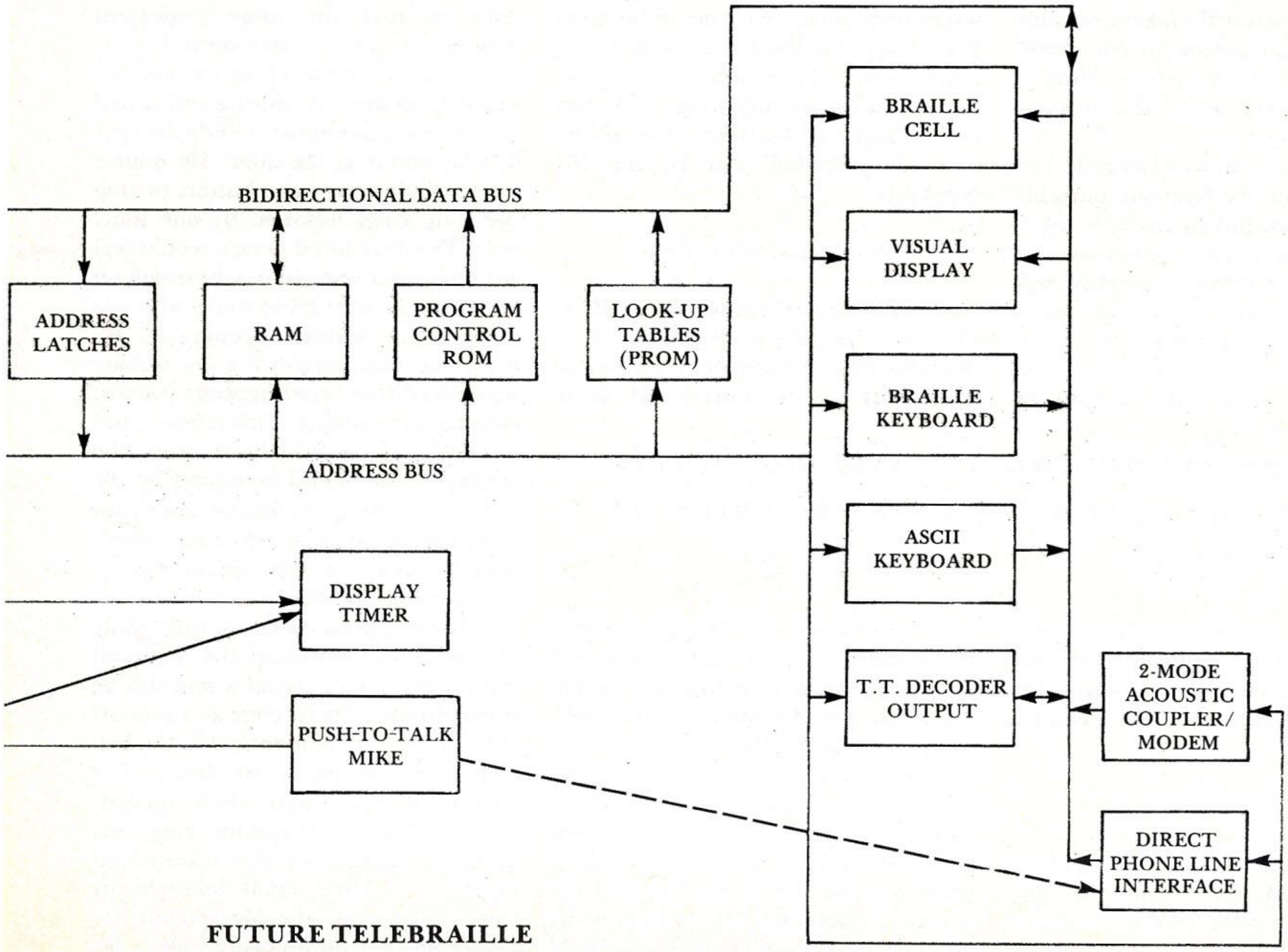
As each Braille character is typed, it is converted into a corresponding audio-frequency-shifted binary data string, which is acoustically coupled to a telephone handset (full duplex, originate and answer modes, and 103 data set compatible). On the receiving end, the handset is acoustically coupled to an identical unit which converts the binary-frequency-shifted string to signals that raise the appropriate pins in the Braille cell. Both letter-for-letter and contracted forms of Braille can be transmitted via the Telebraille.

It is possible to transmit data in both directions simultaneously—a feature which may be more fully utilized at a later time. Right now, however, this ability is utilized to make it possible for one user to send an interrupt signal to the other. In illustration, if user 1 is transmitting too fast, user 2 can press a seventh pushbutton (at the lower left of the Telebraille's panel) and thereby cause user 1's Telebraille to vibrate. Thus, user 1 is made aware that user 2 wants to transmit and can prepare to "receive" the interrupting message.

The Telebraille offers the deaf-blind person a new level of communication. It is packaged in an attache case and can be used with almost any standard telephone handset.

In the future, after the present field testing is complete and small-quantity production has begun, a number of options will be made available. These will most probably include embossed paper tape output, the above-mentioned standard typewriter keyboard, and another, perhaps even more important "add-on"—an electronic converter and coupler to enable the deaf-blind user to contact and communicate with deaf people who use Baudot (five-level) teleprinters in their telephone communication network. Braille will be sent and received by the deaf-blind person. A microprocessor and support circuitry will be used for conversion to and from the serial Baudot code. Provision for sending and receiving ASCII (eight-level teleprinter code) is also planned. Another option will provide for the reception and translation of Touch-Tone-type signals to Braille, to enable a deaf-blind person who can speak and read Braille to communicate with a hearing individual.

Any or all of these supplemental features will enable the deaf-blind user to communicate with a larger number of persons. To communicate with deaf persons using Baudot-coded teleprinters, a different modem design must be used, since the "Deaf TTY Network" makes use of a half-duplex system with mark and space frequencies of 1400 and 1800 hertz respectively. Furthermore, the carrier tone is present only while typing is being done. Clearly there is a major compatibility problem here! To make possible wider use of the Telebraille by the many deaf-blind persons who can speak intelligibly, we have plans for the future modified Telebraille which will enable a deaf-blind person to telephone a hearing person and speak to him. The only sending device the hearing person will



FUTURE TELEBRAILLE
(microprocessor controlled)

require to respond is a pushbutton tone-type telephone or a small portable unit with a self-contained pushbutton dial that can be held over the telephone mouthpiece to send tones back to the caller. For each letter he wished to send, he would have to press two buttons. The first button pressed would indicate the number of the button containing the desired letter and the second would designate the position of the letter on that button. For example, the letter *k* would be sent by pressing button 5 and then button 2, because the *k* is on the fifth button and is in the second position. The letter *y* would be sent by pressing button 9 and then button 3. *That* Telebraille model will electronically translate the two-button sequence to the appropriate Braille character, which will then be displayed to the deaf-blind person as described earlier. Clearly, this will mean that emergency calls, or even social calls, can be made more easily to more people by a number of deaf-blind persons, where the "readout" is a single or multiple letter (that is, a Braille character) or visual display.

In similar fashion, the ASCII/Braille conversion option will make it possible for a deaf-blind person to communicate with a sighted person using a TWX teleprinter—or, for that matter, with a computer.

Until recently, it was virtually impossible to conceive how one portable device could possibly be made to operate in all of these modes. Now, however, it is a possible, although complex, task of developing a microcomputer-based telecommunications system. It is our plan to encode the various conversion tables into read-only memories (ROMs) and to use a microcomputer such as the National Semiconductor SC/MP to carry out the various table look-up, conversion, and control functions.

Emergency calls can be made more easily where the readout is a single letter or visual display.

Wrist-Com

The Wrist-Com is a wireless signaling system for deaf-blind (and deaf) persons. The complete receiver/vibrator is in one small package which is worn on the wrist. For maximum efficiency and economy, two designs of the Wrist-Com have evolved in our Research Department. One is for residential use and one for use in schools or institutions. The former is being developed and prototyped by the National Center's Research Department, while the latter is being developed with the cooperation and assistance of the National Aeronautics and Space Administration, Ames Research Center, and Stanford Research Institute. The designs for both devices have been derived from concepts developed during a joint study with the New York University Deafness Research and Training Center. A list of essential characteristics for wireless vibratory signaling devices was drawn up as a result of this study. It included such features as (1) shock resistance, (2) reasonable range, (3) water resistance, (4) ease of wearing (carrying), (5) limited or no licensing requirements, (6) no antenna or transmitter tuning or adjusting, (7) easy procurement of batteries, (8) ability to send "precoded" signals, and (9) reliability.

In an emergency the Wrist-Com transmitter can trigger an alarm and identify a person in trouble half a mile away.

Residential Wrist-Com

The Residential model of the Wrist-Com will be able to replicate, in vibrations from a receiver worn on the wrist, any signal transmitted to it. The transmitter will be plugged into a standard a.c. wall outlet and will use the household wiring as an antenna. In case of a power failure, self-contained backup battery power will take over and the transmitted signals will still be radiated via household wiring. No Federal Communications Commission (F.C.C.) license will be required, and the transmitter will not require installation other than being plugged into a wall outlet. No field tuning will be required, either. Production receivers will be water and shock resistant.

When the Residential system is connected to an environmental monitoring system, it will be able to send different signals to the receiver for doorbell, telephone, burglar alarm, and fire signals. Furthermore, a Morse code key will be directly connected to the transmitter. This is especially desirable in a household with two deaf-blind persons, where one will be able to call or signal the other without both having to go to a specific location to meet when a simple alert signal is received.

The environmental monitor is actually a major component of the Residential Wrist-Com transmitter package. It is designed to sense its various inputs—doorbell, telephone, burglar, and fire alarm—in a priority order. This is done so that the more important message is always transmitted first. The doorbell is lowest priority; the fire signal is highest. A visiting individual can use a special signal to indicate that it is he who is at the door. He might, for example, press the button to ring two long rings followed by one short ring. The deaf-blind person would feel two long and one short vibrations on his wrist. He would then know who was at the door without opening it. He could, at least, establish a few unique signals to differentiate between friends, merchants making deliveries, and strangers. If the telephone rang, the ringing sound would be sensed by the Telephone Ring Indicator and converted to on-off pulse sequences which would then activate the environmental monitor and the Wrist-Com transmitter. If the phone were to ring while someone was pressing the doorbell button, the phone signal would still be transmitted to the receiver as an on-off vibration sequence identical to the bell ring-silence sequence we hear. If a window or door were then opened, even during the telephone ring, the monitor would alter the transmitted signal to a long signal followed by three short ones, the Morse code for the letter *B* (dash-dot-dot-dot), for burglar alarm. If something in the apartment were to begin to burn, a combustion detector would send that message to the environmental monitor and the Morse code for the letter *F* (dot-dot-dash-dot) would be transmitted. Thus each higher priority signal would be transmitted even though other signals were in progress.

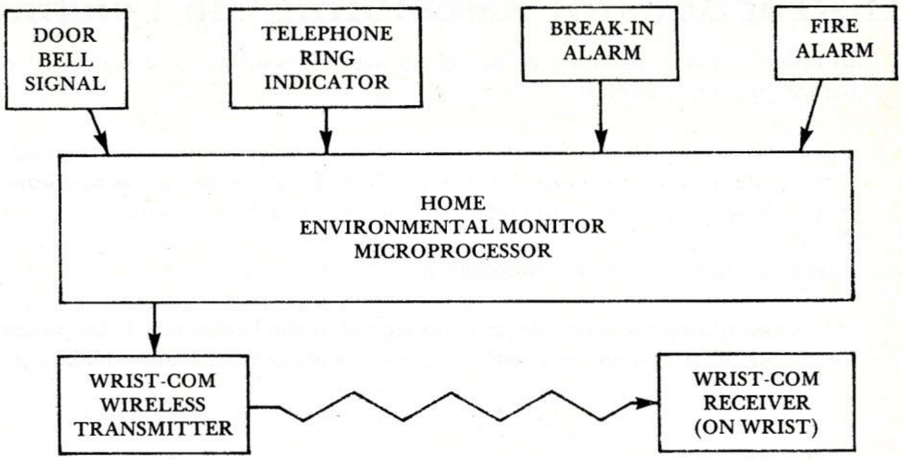
We are designing the monitor in such a way that the type and priority of signals transmitted can be easily customized to suit individual needs. Even the speed of signal transmission will be adjustable.

Quite obviously, a microprocessor can be integrated into the environmental monitor too. Small- and medium-scale CMOS integrated circuits have been used in the prototype. It is probable that a microprocessor will be used to replace most of the integrated circuits, since labor would be reduced significantly and it would enable us to customize output signals and other functions to each user's needs at minimum expense.

The first Residential Wrist-Com receivers and transmitters will be built during the next twelve months. These first units will be put to temporary use at the Helen Keller National Center, thereby enabling us to evaluate them and make any necessary improvements before distribution to other users.

Institutional Wrist-Com

The Institutional model of the Wrist-Com is far more complex than the Residential model. In this system, the controls on the master transmitter console will enable the operator to select an individual, a group, or all Wrist-Com receivers for a message. The receivers are quite sophisticated, using microcomputer circuitry to decode transmitted impulses, signal the wearer that a message has been transmitted (the entire unit vibrates), and then repeatedly send a Morse code letter to a vibratory "finger pad" on the top of the Wrist-Com until the unit is reset by the wearer. When reset, a signal is transmitted to the master console to indicate reception of the message. In an emergency, the same transmitter can trigger an alarm and identify the person in trouble. The self-contained microcomputer will also be used to test the status and proper operation of the receiver. It will have a range of one-half mile or more.


DOOR BELL SIGNAL
TELEPHONE RING INDICATOR
BREAK-IN ALARM
FIRE ALARM
HOME ENVIRONMENTAL MONITOR MICROPROCESSOR
WRIST-COM WIRELESS TRANSMITTER
WRIST-COM RECEIVER (ON WRIST)

1. CODED DOORBELL SIGNAL POSSIBLE
2. TELEPHONE—CYCLIC ON/OFF VIBRATION
3. _ . . . BREAK-IN
4. . . _ . FIRE

BLOCK DIAGRAM OF THE RESIDENTIAL WRIST-COM AS IT PRESENTLY EXISTS IN PROTOTYPE FORM

If licensing permits, complete Morse code messages can be transmitted to a given individual or group. Where this is not possible, single Morse letters will be used to represent different messages. Unfortunately, the Institutional Wrist-Com is still several years from completion. When it is completed, however, it will replace the more basic units presently being used at our Sands Point facility.

Both the Telebraille and Residential Wrist-Com are rather complex units, and they involve machine work as well as electronic assembly. The Telephone Ring Indicator and the Tactile Speech Indicator, however, are less complex, and they can be easily built by the electronics hobbyist.

TEMPORARY SIGNALING SYSTEM

In order to provide emergency signaling to our most severely handicapped deaf-blind clients immediately, we developed a temporary alert system. We modified Motorola Pageboy II receivers so that they would all respond to the same carrier-plus tone signal with a steady vibration which continued for the duration of the signal. We send class period start-and-end (seven seconds on) and fire alarm (one second on—one second off—one second on—one second off, etc.) signals. The use of these receivers with the "battery-saver" option has meant that we only have to replace the one "N" size, 1.5-volt mercury battery every three or four weeks. Signals are sent and received on a special NASA radio frequency assigned to the Wrist-Com project.

For those who want to utilize their technical skills to help deaf-blind and/or deaf persons, the following section contains a schematic diagram, parts list, circuit board layout, board stuffing guide, and detailed step-by-step instructions for the construction of the Telephone Ring Indicator. (A construction project for the Tactile Speech Indicator will be presented in a future article.)

If you have the background, interest, and facilities to manufacture special devices for deaf-blind people, or for people with *any* handicap, please remember that it is imperative that a number of deaf-blind consumers be involved in the project as early as possible. By enlisting their help, the likelihood is very great that the end product will actually be usable by deaf-blind persons. Moreover, if there is some need for individual modifications of the device, provision to facilitate customizing changes can be designed into the unit. This cannot be emphasized too strongly! Any aid or device, whether intended for use by a single handicapped individual, or planned for a group of handicapped persons, must be developed *with* consumer consultation throughout the project. Otherwise, there is a great risk that the well-intentioned designer will produce what may be a technological marvel to most people, but a useless novelty to the actual consumer. ▼

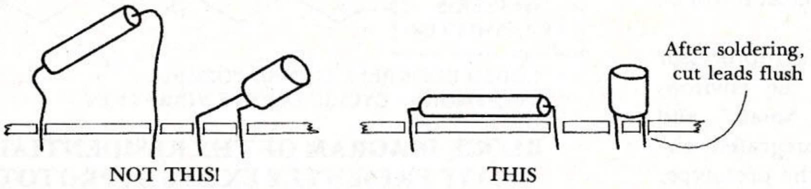
Instructions for Assembling the Telephone Ring Indicator

In order to make assembly as straightforward as possible, it is suggested that you read through the instructions before you begin assembly.

Note: All components are mounted on the same side of the printed circuit board as the potentiometer. To avoid errors, solder each component as it is installed. A *40-* or *50-watt pencil iron* is best. Be sure to use *resin core* solder only. Always insert the components so that they touch the board.

Check off each step as you complete it.

When assembling the unit, please make sure that the bodies of all components, except transistors, are lying against the circuit board wherever possible. Trim component leads after soldering.



☐ 1. Place the circuit board in front of you in such a way that the potentiometer is towards you.

☐ 2. Straighten out the wire leads of the relay (RY1) (Radio Shack #275-003 or equivalent 12-volt d.c. coil, 1200 ohms, 10 ma).

☐ 3. Hold the relay (RY1) bottom side up with the 3-lead side on your right.

In the next steps you will cut the relay leads to different lengths to make mounting in the board easier.

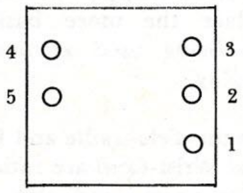


Figure 1
RY1 (bottom view)

☐ 4. *Referring to figure 1,* cut one inch from lead 1.

☐ 5. Cut one-half inch from leads 2 and 5.

☐ 6. Now mount the relay, inserting leads 3 and 4 in the holes labeled 3 and 4 in figure 2 (see p. 85). Don't push the leads all the way in yet.

☐ 7. Now insert leads 2 and 5.

☐ 8. Finally, insert lead 1 and gently pull the wires until the relay is snug against the board.

☐ 9. Turn the board upside down and solder the five wires. Be sure that the relay is held tight against the board before you solder it in place.

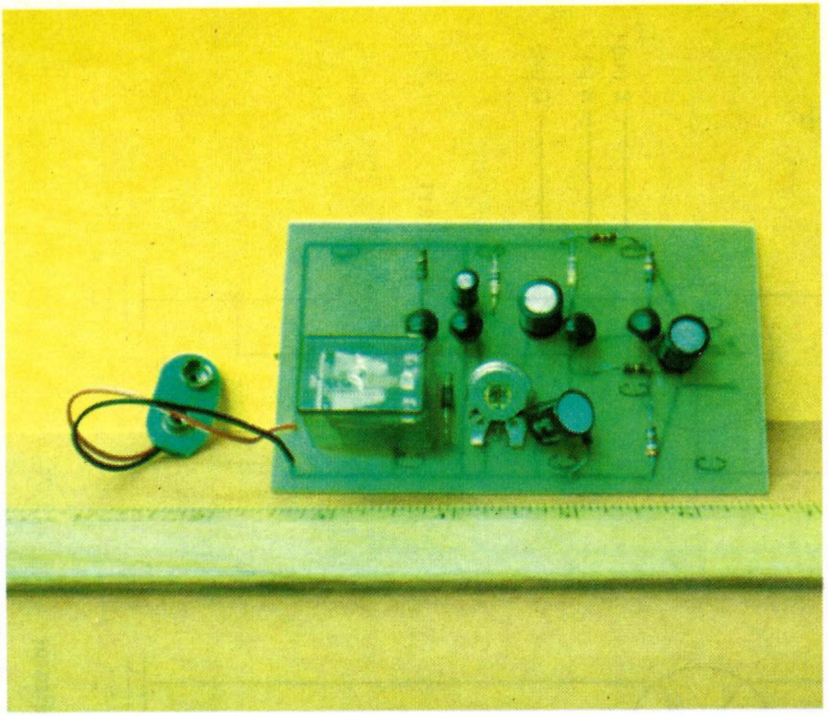
The capacitors are now mounted. In some cases, you may find that substitutions have to be made. For instance, a 22-mf capacitor may replace a 25-mf one. You can use tantalum capacitors if you have them. Don't worry.

In all the steps below, refer to figure 2 and the schematic.

☐ 10. Reorient the board as in step 1.

☐ 11. Insert C1 (15 mf). Be sure that the negative (—) lead is closest to the right side of the board. Solder the two leads.

☐ 12. In like manner, insert C2, C3, C4, and C5, making sure that the negative (—) mark is as shown in figure 2.



In the top view of the assembled Telephone Ring Indicator printed circuit board, the relay (RY1) is at the lower left and the potentiometer (R9) is to its right.

In the following steps, the resistors (all ¼ watt, 5 percent) are mounted and soldered (refer to figure 2 and the schematic).

☐ 13. Insert 56K-ohm resistor (green-blue-orange) at R1 and solder.

☐ 14. Insert 390-ohm resistor (orange-white-brown) at R2 and solder.

☐ 15. Insert 6.8K-ohm resistor (blue-gray-red) at R3 and solder.

☐ 16. Insert 1.5K-ohm resistor (brown-green-red) at R4 and solder.

☐ 17. Insert 1.0K-ohm resistor (brown-black-red) at R5 and solder.

☐ 18. Insert 75-ohm resistor (violet-green-black) at R6 and solder.

☐ 19. Insert 100K-ohm resistor (brown-black-yellow) at R7 and solder.

☐ 20. Insert 82K-ohm resistor (gray-red-orange) at R8 and solder.

☐ 21. Now locate D1. It may be either glass or black epoxy. It will have a single band around one end.

☐ 22. Insert D1 next to RY1 so that the end with the band (+) is nearest Q3 and Q4.

The four transistors are mounted next. Note that Q4 is a 2N3638A, while Q1, Q2, and Q3 are 2N3643. Try to keep the bodies of the transistors one-quarter to one-half inch above the board. Transistors will match holes only one way, when inserted from the *top* side of the board (no printed circuit on this side).

☐ 23. Insert and solder Q1 (2N3643).

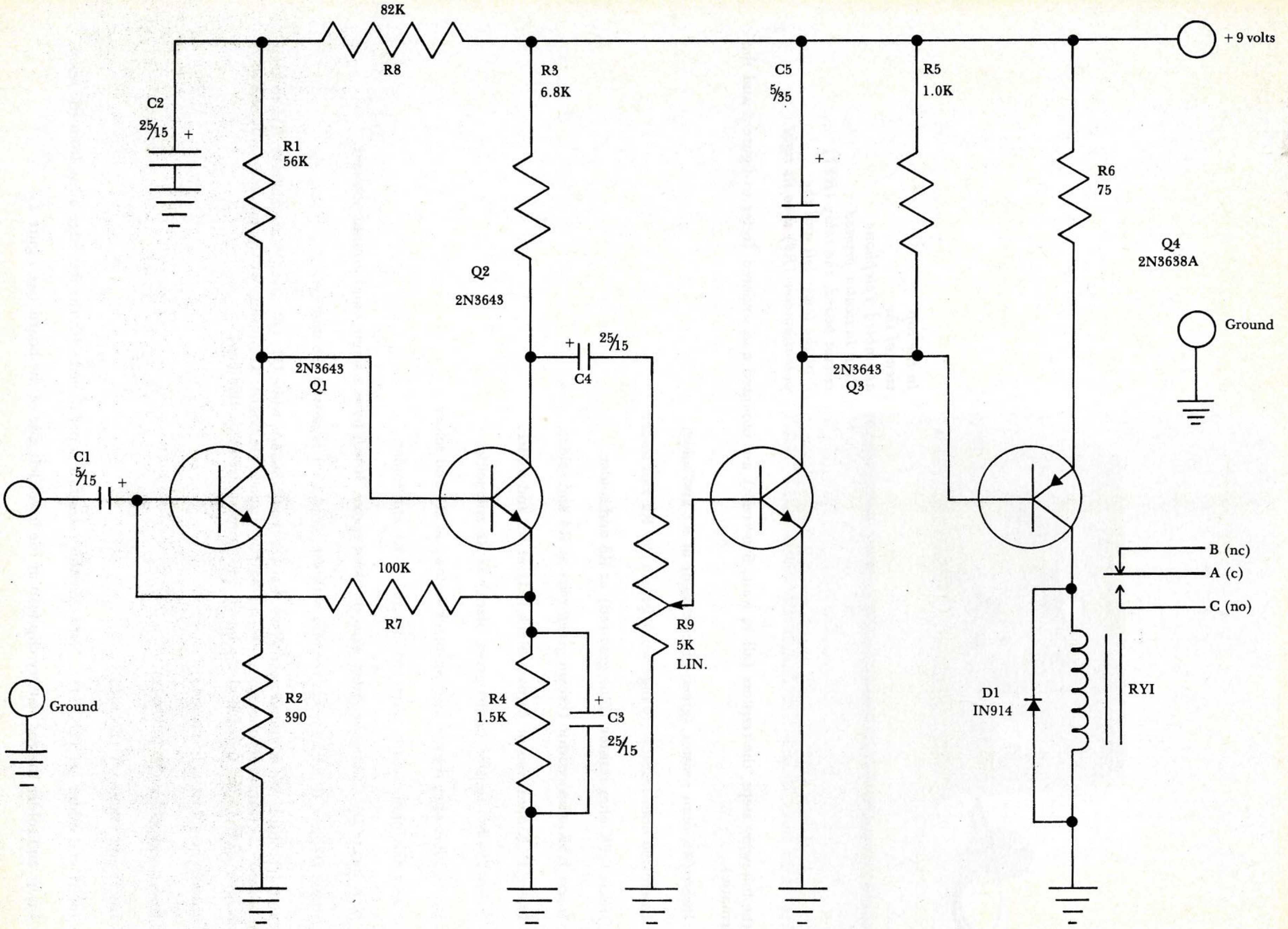
☐ 24. Insert and solder Q2 (2N3643).

☐ 25. Insert and solder Q3 (2N3643).

☐ 26. Insert and solder Q4 (2N3643). Note that this transistor is positioned with the flat edge *away* from the relay.

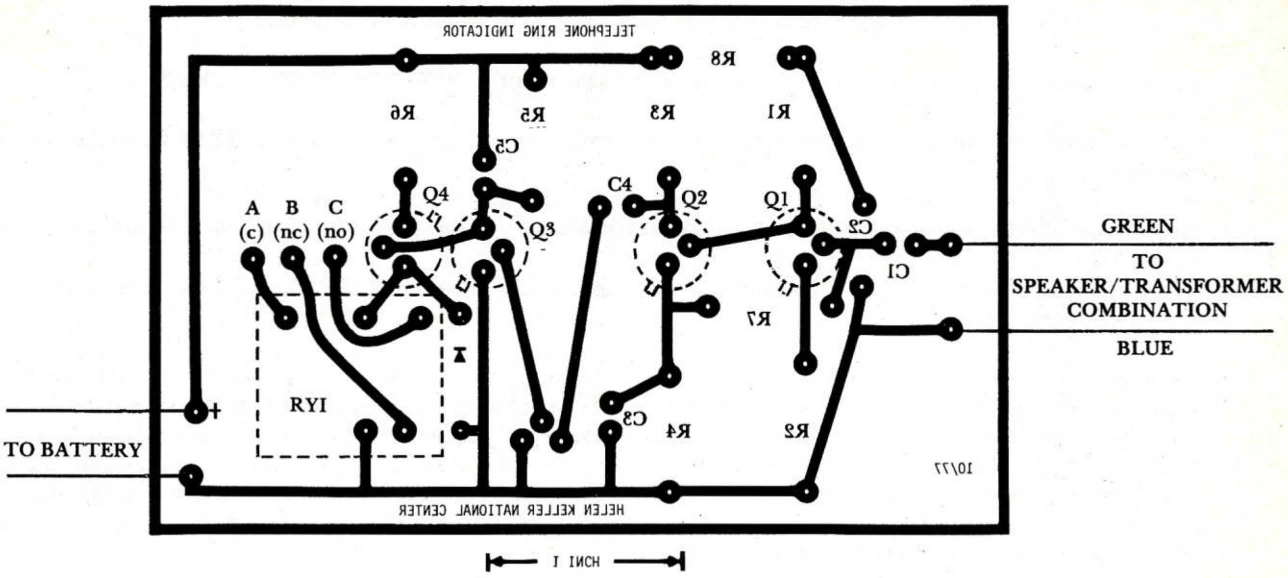
☐ 27. Insert and solder the red battery clip lead at the lower left side of the board (see figure 2).

☐ 28. Insert and solder the black battery clip lead at the lower left corner of the board (see figure 2).

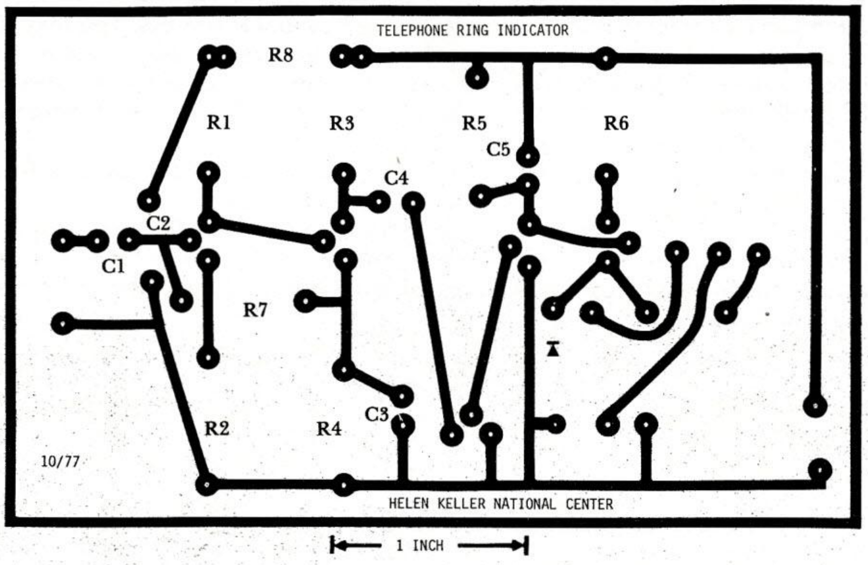

82K
R8
+ 9 volts
C2
25/15
+
R1
56K
R3
6.8K
C5
5/35
+
R5
1.0K
R6
75
Q4
2N3638A
Q2
2N3643
Ground
+
25/15
C4
2N3643
Q1
2N3643
Q3
C1
5/15
+
100K
R7
B (nc)
A (c)
C (no)
R9
5K
LIN.
Ground
R2
390
R4
1.5K
+
C3
25/15
D1
IN914
RYI

Schematic Diagram of Telephone Ring Indicator

Figure 2
Board Stuffing Guide for Constructing the Telephone Ring Indicator


TELEPHONE RING INDICATOR
R8
R6
R5
R3
R1
C5
C4
Q4
Q3
Q2
Q1
C2
C1
A B C
(c) (nc) (no)
RYI
+
R7
C3
R4
R2
10/77
HELEN KELLER NATIONAL CENTER
1 INCH
GREEN
TO SPEAKER/TRANSFORMER COMBINATION
BLUE
TO BATTERY

CIRCUIT BOARD LAYOUT FOR TELEPHONE RING INDICATOR


TELEPHONE RING INDICATOR
R8
R1
R3
R5
R6
C5
C4
C2
C1
R7
C3
R2
R4
10/77
HELEN KELLER NATIONAL CENTER
1 INCH

Note:

If you decide to construct one or more Telephone Ring Indicators for deaf-blind persons, please inform the Research Department of the Helen Keller National Center of your plans. We have some already etched and drilled circuit boards and parts which we can send to you. When you complete construction and return the unit to us, we will make it available to a deaf-blind person. If you wish to construct a Telephone Ring Indicator for a deaf-blind or deaf person you know, please let us know also.

List of Electronic Parts for Telephone Ring Indicator

C1, C5	5-mf 15-volt electrolytic or tantalum capacitors
C2, C3, C4	25-mf 15-volt electrolytic or tantalum capacitors
D1	1N914 (or equivalent)
Q1, Q2, Q3	2N3643 (or equivalent)
Q4	2N3638A (or equivalent)
R1	56K ohms, ¼ watt, 5%
R2	390 ohms, ¼ watt, 5%
R3	6.8K ohms, ¼ watt, 5%
R4	1.5K ohms, ¼ watt, 5%
R5	1.0K ohms, ¼ watt, 5%
R6	75 ohms, ¼ watt, 5%
R7	100K ohms, ¼ watt, 5%
R8	82K ohms, ¼ watt, 5%
R9	5K linear taper potentiometer (Mallory MTC53L4 or equivalent)
RY1	SPDT mini relay (Radio Shack #275-003 or equivalent)
T1	1000 ohms center tapped to 8 ohms.

☐ 29. The blue and green leads from the speaker/transformer combination may be temporarily connected to the two input pads at the middle right edge of the board.

This completes the assembly of the electronics portion of the Telephone Ring Indicator.

To test for proper operation, connect a 9-volt battery (equivalent to 2U6, 216, VS323, or 006P).

☐ 30. Using a small screwdriver (inserted into center opening), rotate the potentiometer (R9) to its full clockwise position.

☐ 31. Connect an ohmmeter, or a series-connected lamp and battery, to relay terminals labeled in figure 2 as A and C.

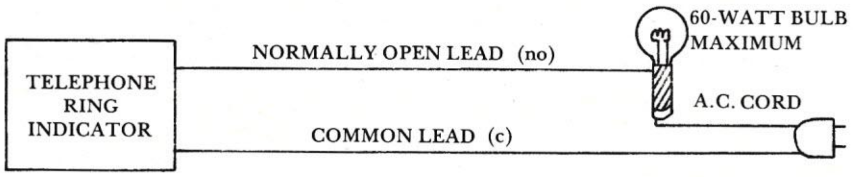
☐ 32. Whistle or yell about three to ten inches from the speaker. The meter should deflect, or the lamp light, for the duration of the sound.

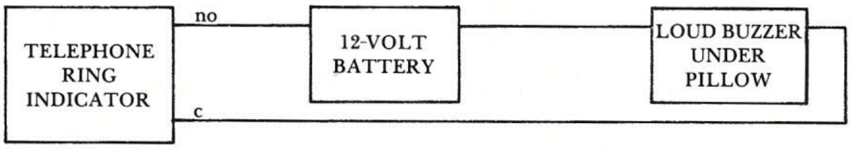
☐ 33. Once you are satisfied that the unit is operating properly, disconnect the battery and the speaker/transformer combination from the printed circuit board and install the board in an appropriate case. Be sure to make a large enough opening in the top of the case for the microphone. If you wish, the sensitivity control can be removed from the circuit board and a bushing-mount potentiometer of the same value installed through the side of the box. This isn't necessary, however, since the control is usually not adjusted once it is set for a particular phone ring. Use a box large enough to sit under the base of the phone, or try to substitute a microphone which can be cemented to the side or base of the phone.

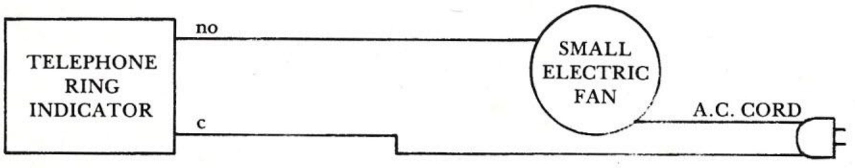
Note: If you wish, you can substitute a dynamic microphone element, with a 10,000-ohm impedance, for the loudspeaker/transformer combination. Be aware, however, of the fact that the two- or three-inch loudspeaker and 1000-ohm-to-voice coil transformer are more readily available and will probably cost less. You may find it convenient to substitute a relay you already own for the one specified. If you do, be sure you use a sensitive one with contacts capable of handling at least one amp. Enough open real estate has been left in the relay area for you to use a physically larger relay by cutting back on the lands somewhat and drilling new lead holes. In some applications, you may wish to substitute an optic isolator for the relay in interface directly with TTL or CMOS logic, or even with the keying line of a radio transmitter. (Watch your voltage level and polarity, though, if you use the optic isolator!)

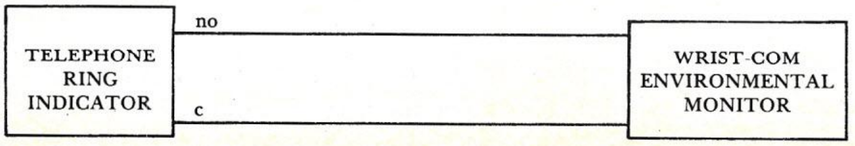
A 9-volt d.c. adapter may be used in place of the battery for maximum long-term reliability. This is what you *should* elect to use if you are constructing the device for a deaf-blind or deaf person who will be depending upon it. It is most desirable to use a plastic box with no metal portruding, to reduce shock hazard. Be sure, however, to use a sturdy box so it doesn't break if dropped.

Below are a few sketches of how the Telephone Ring Indicator might be connected to various signaling systems.



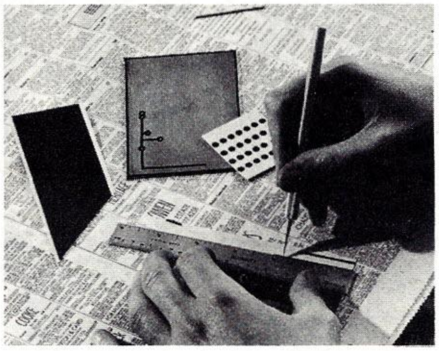


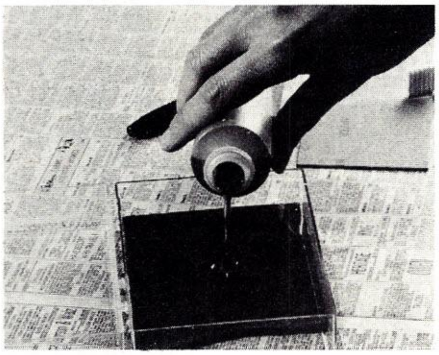


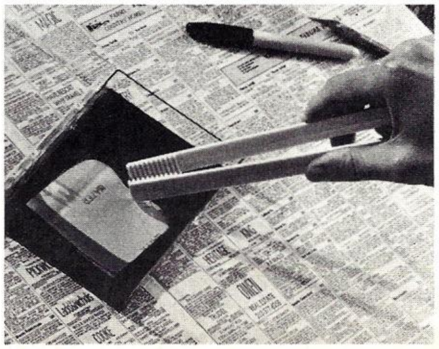


FIRST-TIMER'S GUIDE TO CIRCUIT BOARD ETCHING

by Thorn Veblen







The advent of printed circuit board kits has enabled even the beginner to turn out a first-class professional-looking finished product. No longer need one be plagued by a rat's nest of wires and double soldering.

Kits are available at most electronics stores and include the blank circuit boards, resist-ink pens, the liquid etchant, ink solvent, and usually even a one-sixteenth-inch drill bit for making the holes in which the component leads will be inserted prior to soldering. The newer kits also include pressure-sensitive circuit lines and ringlets, that can be peeled off and applied to the board forming the circuit. These are easier and much neater to work with than the resist-ink pens.

The first step in kit-making is to make sure you start with a clean board. The last circuit board I had arrived with a greasy palm print right in the center. Another board was covered with a thin lacquer coating to prevent oxidation. In case your kit comes through with a less-than-sparkling copper face, give the board a light scouring under running water. Use very fine steel wool. If all the dirt is not removed, you may end up with the etchant failing to dissolve all the extra copper and leaving some very strange circuit patterns indeed.

Place the cleaned board on a firm table copper-side up. On top of it, lay some carbon paper with the copying side facing the copper. Now, on top of this, place the circuit diagram (such as the one in the preceding article by Dr. Kruger). Trace the circuit diagram with a ball-point pen. *Make sure you cover every line and hole.* Once the board is etched, you can't replace the copper to make up for a connection you missed.

After the diagram has been transferred onto the copper, apply the peel-off tape lines and appropriate circles for the component leads. Do not peel off a long piece of tape and lay it directly onto the circuit board for cutting to size; doing so can lead to cutting through the relatively thin copper layer and breaking the circuit when you trim. Instead, measure and cut the strips to size as you peel them off their original backing.

Once the completed circuit is in place, burnish all the tape lines by pressing them down firmly with the back of a spoon. This will assure that tape covers the copper completely. It it fails to do so, the circuit lines will be ragged and sloppy.

Pour some etchant into a plastic tray. (The etchant is very corrosive, so avoid contact with metal—also avoid contact with skin since it can cause yellow nicotine-like stains that last for weeks.) Now slide the circuit board copper-side down into the liquid. Let it stay there for twenty minutes or until all the visible copper has been removed leaving only the copper traces beneath your masking. Then remove the circuit board, using plastic tongs, and wash it under cold running water for a few minutes.

Now drill the appropriate holes for the component leads. When you solder the component leads in place through these holes, use a low-heat iron and be careful not to lift the copper traces. After you're finished, clean the board with the resist-ink solvent. As a last step, spray the completed circuit board with a protective coating of polyurethane to prevent oxidation.

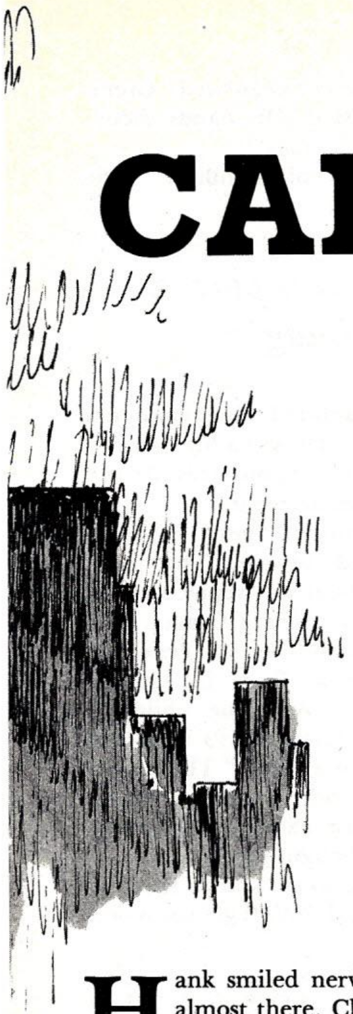
Congratulations! You're all done. ▼

Photographed by Thomas Hall

CALL EXIT

A Short Story by

Frederick W. Chesson



NEW YORK, May 5 (AUP)—Gamewell P. Gremshaw, forty-two, founder and former president of G & G Industries, died early yesterday afternoon when he suffered a fatal heart attack at the entrance of the 57th Street station of the Westchester subway.

Once known as "The Computer King," Gremshaw slipped into obscurity when his once unprecedented systems were rendered obsolete by newer techniques. His demise in the very shadow of his old corporate headquarters, now known as Global General, was followed by a statement from that company that Gremshaw had been invited to a reception for the presentation of an award in gratitude for his services to the art.

Funeral arrangements are in the charge of the data industry's "Old Programmers' Mission" on the Bowery, where Gremshaw had lived for several years.

Hank smiled nervously. "We're almost there, Chief."

"I know." Gamewell Gremshaw nodded. "I used to ride this line every day, when we lived up in Quaker Ridge. That was when the Second Avenue Line was extended up to White Plains. And that meant the old Westchester Railroad was reborn. Ah, rebirth. You know, we could be up there in . . . thirty minutes or so." He shut his eyes tightly. "But, Hank, there wouldn't be anyone there anymore. Not anymore . . . ."

"Steady, Chief." Hank fumbled for the little bottle in his jacket and opened it. "Here. After the big meeting, everything will be just like old times again. Didn't they promise you so?"

"To old times." Gremshaw clutched at the bottle and tilted it.

"Easy, Chief. That's cognac. Real high-class stuff. You got to be in top shape when you talk to all those big shots at G & G."

"Phonies!" Gremshaw spat. "None of them could have made even Junior Programmer in the old days. Not one of them can write a FORTRAN program, or make a flowsheet. Not one systems analyst in that whole Global General crowd. That's why they want me back now!"

"Easy, Chief." Hank retrieved the nearly empty bottle. "Remember your old ticker. The Mission doc said to watch your heart."

"I know. The old Master Oscillator is all out of synch. Someday it'll drop one bit too many, and that'll be the end of the file, Hank. That'll be CALL EXIT for the Gremshaw program."

"Think positive, Chief. Today, it's . . . OPEN NEW FILE for you."

Gremshaw patted Hank solemnly on his threadbare jacket. "And for you, too. You're coming up to the top with me. You're the only one on the Row who knew I wouldn't stay a bum forever . . . ."

Hank smiled. "Oh, I always knew you had real class."

"Thanks, pal. You're going to be the only executive assistant who knows what it's like to clean windshields at the Canal Street stop for enough tips to share a bottle of Big Byte bourbon. Or . . . . Or to pick out enough transistors from old computer boards in Benny's Surplus Center to earn enough for another night at the Mission when you're too sick or too proud to ask for a free bed. Ah, Hank, we're here. Fifty-seventh Street, and G & G!"

"Take it easy, Chief." Hank helped him from the car.

Gremshaw peered about the platform, swaying as the train gusted out of the station in a blast of stale air. "I always used to buy my *Wall Street Journal* here, or was it the *Times?*" He approached the newsstand. "The *Wall Street Journal,* please."

The attendant eyed him. "Okay, Pops. But for sleeping on a bench, the *Post* has more body to it." He turned to a loudspeaker grille. "Data. One *Wall.* Cash. Data out."

"I have a credit card . . . at home," Gremshaw said.

"Sure, Pops." A slot spewed out the newspaper and the man accepted Hank's coin. "If you make a killing today, buy me a hunnert shares, huh?"

Gremshaw studied the unfamiliar front page. "The *Journal* seems to have gone tabloid. Just like the old *Mirror.*"

"You mean the *News,* Chief." Hank grinned. He looked up at a clock, part of an animated birth-control-product advertisement. "We don't want to be late for our big meeting, I mean yours."

"Let them wait for us!" Gremshaw folded the newspaper. "Perhaps I should have acquired a new wardrobe."

"You look plenty spiffy to me," Hank said approvingly. "The hock shop kept real good care of your old duds."

Gremshaw looked at himself in the mirror of a vending machine. He avoided the intrusion of the omnipresent speaker grille, knowing its insolent

Illustrated by Steve Gerling

silence to the pleas and curses of the uncredited. "Voice control," he thought. "I remember when Richardi and Clanton left G & G to start Vocatronics. I sold the patent outright to them for a mere $5,000! They bought out Datavox and merged with Computer-Confab. Then Dynamcometrics took over. G & G was shaky then. They offered to make me Exec VP. I told them to go to hell. A week later, G & G became Global General and I was out. I wonder where I would be today if. . . ."

"Hey Gravy! Whatcha doin' way up here?" A familiar figure from the Mission shuffled over to embrace him. "What you an' Hank doin' in this part a town?"

"Hi, Mushy." Hank moved to separate them. "What are *you* doing so far off the Row?"

"I carry important messages around for big people." He waved a sheaf of small papers.

Hank shrugged. "Policy slips. Small-time stuff, ever since Credit Card Casino was legalized. Definitely small-time stuff."

"Look who's talkin'! I bet I make more in one morning than you and Gravy-Eyes together make in a whole week." He gave a snaggle-tooth grin and pointed at Gremshaw. "I know you! You're Gravy-Eyes Gremlin, the Computer King! Nothin' but a bum now!"

"Leave him alone, Mushy." Hank eyed an approaching Transit patrolman. "Let's go, Chief."

"Don't rush me." Gremshaw started for the stairs. "But maybe he's right. What does Global General see in me? I'm no more use to them than an old overpunched IBM card. A worn-out tape." He paused for breath on the landing, letting a small mob of schoolboys hurtle down the steps. "I think they just want my name. For a fancy figurehead, a letterhead. For another dummy corporation.

The old Master Oscillator is all out of synch. Someday it'll drop one bit too many and that'll be the end of the file.

"Don't I know that COBOL and FORTRAN were killed by VOCAL, Voice-Oriented Computer-Allocated Language, Hank?" Gremshaw looked up the stairwell at light streaming down through sparkling dust particles. "But now voice terminals are going to be replaced by visuals! It's in the wind. I'll outfox them all!"

The great clarity of the light was replaced by a shadow world of shifting images.

"Take it easy, Chief," Hank said with some alarm. "Maybe we should come back tomorrow."

"No, Hank, it's all the way today. Right to the end of the tape." Gremshaw pushed past Hank and took the remaining steps two at a time. "I'll get the best brains in the visuals field. Put together another system that they can't compete with. Buy 'em out or force 'em out. Global General will become G & G again!" He leaped the last steps to the top of the stairs. Above him loomed the familiar bulk of the G & G skyscraper. "My G & G!" The sun shone in great clarity. Then the clarity became a thing of pain and he lurched forwards into nothingness.

The two officers arrived as Hank was sliding the folded newspaper under the fallen man's head. "It's his heart, officer."

"Let us decide, Mac." One bent close. "Smells like bottle fatigue to me."

"Medicinal brandy," Hank said. "His old ticker gave out when he ran up the last flight of stairs. He was going to a business meeting at the G & G Building across the street."

"Oh, sure." The other policeman raised his shoulder-slung walkie-talkie. "Twenty-five A-seven here. Data in. Medic rep. Heart attack or alcoholic stupor. Victim is white, male, elderly, indigent. No evident credit number. Data out."

"Say, he's got one after all, Charlie." The kneeling officer examined a battered plastic card from the musty overcoat pocket. "Triple zero, two-two-six-five. Hey, that's a damn low number."

"That's *Mister* Gamewell Gremshaw," Hank said. His hands shook. "Gamewell Gremshaw!"

"Sounds sort of familiar." The officer stood up.

Charlie transmitted this and secured his radio. "Sure. He was a big name in computers once. And now this. An old wreck in a good overcoat."

"Brooks Brothers' best English," Gremshaw said weakly. The great clarity of the light was replaced by a shadow world of shifting images. The Custom Shop at Brooks. Images of the executive suite at G & G. Quaker Ridge. Sharon. And the children. "Core-storage dump, that's what this is. Sharon, are you there?" The images flickered past, but would not be recalled for a last caress. "Destructive readout," he thought sadly.

"We're going to get you to the hospital," Hank said. "We'll get you a new heart, Chief."

Gremshaw looked up at him. "Too late, Hank. It's CLOSE ALL FILES for me. Time to CALL EXIT."

"Call who?" asked Charlie.

"Call the wagon. I think this one's going to be DOA." Ed gestured to some hovering spectators. "Okay, let's keep moving, folks."

Hank stood in the phone booth, kicking aside the fallout of newspapers, beercans, and dog dung. The video plate of the phone was nearly obscured by layers of lipstick and spittle. He inserted an extra coin and the defaced surface was replaced by a clean area. "That's one time when the Human Factors boys forgot to take human nature into account." The plate swirled into color. Then the connection was made in a confidential code of tartan plaids, which confirmed the identity of the party without revealing it.

"Bad news, Chief. . . ." Hank began. "Oh, you saw the whole show from the doorman's remote TV pickup. Well, then you know Rothback will just have to come up with a new front for your corporate reorganization." Confidential investigator Harrold Killington, known of late as Killer Killie on the Row, nodded at the viewplate. "It had better be someone from the old group who hasn't fallen quite as far as Gremshaw. I mean, can you imagine being called Gravy-Eyes Gremlin?"

Killington, specialist in the growth spasms of unmentionable corporate colossi, smiled. "Did you know that old Gremshaw was wise that VOCAL had about run its course? Just before he keeled over, he predicted that Visual Credit ID would be the new wave. Just like he knew the tab for converting Greater New York down to the last kilobuck and the last megabit of storage."

He pulled the door tighter as a large woman with packages demanded immediate occupancy. "I got to like him a lot. You know, we'd sit around with a bottle of muscatel, from salvaging parts off old printed circuit boards. And we'd talk right along in those old computer languages like COBOL and FORTRAN. All obsolete, like him, but when you'd think how they'd built the data industry together. . . ." Killington turned, still in the personality of Killie. "Innaminute, lady! Whatcha want to do in here, take a piss or something?" He scratched his stubbled chin and regarded the anonymous screen. "That's the real hell of it, Chief. Too soon we all get obsolete these days. Obsolescence is worse than death."

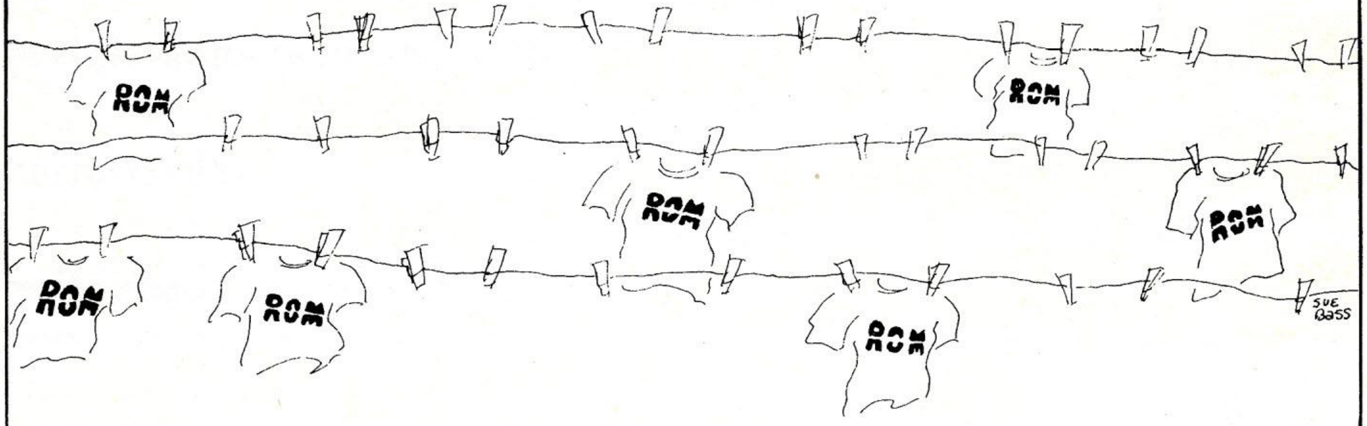
> *The plate swirled into color; the connection was made in a confidential code of tartan plaids.*

Killington fended off the woman temporarily with an obscene gesture. "Well, this sudden death clause ends my part of the contract. Going to take a nice long vacation before the next job. Just send the check to my Nassau bank account. Well, it's been interesting to work for you, Mr. . . . Chief. Well, goodbye. Data out. Oh, yes . . . and CALL EXIT."

Killington toasted the scowling woman with the cognac's last fiery ounce and headed back to the subway entrance.

He noted the newspaper still neatly folded on the sidewalk where it had cushioned Gremshaw's head. "Poor old Gravy-Eyes. His last *Journal.*" He retrieved the paper, dusted off a heel mark, and descended into the stagnant updraft.

A long ride to nowhere in particular would help to taper off from the job's demand of total role identity. The funeral could be attended in the personality of either Killington or Killer Killie, depending on where it was held. Let *them,* or their computers, make the decision. In the meantime, a ride into limbo. North, into Westchester. Perhaps even up as far as Quaker Ridge. ▼



Get your ROM T-shirt now!

Order from:
ROM Publications Corp.
Route 97
Hampton, CT 06247

Please ship to:
Name __________
Address __________
City __________ **State** __________ **Zip** __________

☐S Qty. ______
☐M Qty. ______
☐L Qty. ______ **☐Child S Qty.** ______
☐XL Qty. ______ **☐Child L Qty.** ______
☐Check or money order enclosed. **$5 ppd.**
☐Master Charge ☐BankAmericard **2 for $9 ppd.**
Exp. date **Card#** __________

Please allow 4 to 6 weeks for delivery.

You're curious enough to read ROM now, you'll be curious enough to read ROM every month. So subscribe!

SAVE $ 7.00 **over the single-copy price with a one-year subscription**

SAVE $20.00 **over the single-copy price with a two-year subscription**

SAVE $33.00 **over the single-copy price with a three-year subscription**

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97
Hampton, CT 06247

Name ______________________________

Address ______________________________

City ______________ **State** ______________ **Zip** ______________

United States: ☐**One year $15** ☐**Two years $28** ☐**Three years $39**
Canada and Mexico: Please add $2 per year additional postage
Europe and South America: Please add $12 per year additional postage
All other continents: Please add $24 per year additional postage
☐**Check or money order enclosed** ☐**Master Charge** ☐**BankAmericard**

GUARANTEE:
If not satisfied with **ROM** at any time, let us know. We'll cancel your subscription and mail you a full refund on all copies still due you.

Please allow 4-6 weeks for delivery. **Exp. date** ______ **Card#** ______________

If you wish to be billed, please use either Master Charge or BankAmericard. Direct billing by the publisher is the single largest cause of subscription service problems. With today's postage rates, it's also very expensive for you as well as for ROM.

Legal ROMifications

WARRANTIES?

by Peter Feilbogen, attorney at law

The law has been wrestling with the consumer, the manufacturer, and the retailer for hundreds of years. It still has problems. The crux of the matter was put to me by one of my clients this way: "The money I paid them was perfect—I expected the product to be as good." It's very difficult to argue with such logic.

On a small-ticket item, the failure of the product to perform to specifications is an irritant like the feeling one gets when one has been cheated. On a large-ticket item, it can be a disaster. When you spend a good sum of money, be it for a home computer or a car, and the product is deficient, you can be in for real difficulties.

In dealing with this subject, as with most subjects, there is state law and federal law to consider. The federal law is really an attempt to deal with the automobile without specifically naming it. The bill centers on the *written guarantee.* If a written guarantee is made available by the manufacturer, it must be shown to the consumer before purchase of the product and must be clear and precise.

There are gradations of warranties from full to as limited as the manufacturer sees fit. There is no obligation on the manufacturer to offer a warranty. If a full warranty is offered, then a refund must be made available or there must be a replacement of defective products after reasonable attempts at repair.

Once the law was passed, it fell upon the bureaucracy of the Federal Trade Commission to set forth standards of reasonableness for manufacturers' attempts to repair products before a replacement was mandated. To date the work on such standards has not been completed.

Consumer groups which had hoped that the legislation would help automobile buyers are disappointed in the bill because the three largest automobile manufacturers have not offered full warranties. It is still too early to assess how this law has aided consumers in purchasing other articles, such as computers, except to say that the benefits, if any, are much slower in coming.

On the state level, there exists a set of statutes called the Uniform Commercial Code. New York has adopted this code, as have most other states. The UCC deals with warranties on a different basis than that of the federal law. It is a basis for the day-to-day transactions of commerce and goes well beyond the mere requirement to exhibit a warranty. The UCC codifies the methods by which warranties are created.

1. Express warranties by the seller are created as follows:
    - a. Any affirmation of fact or promise made by the seller to the buyer which relates to the goods and becomes part of the basis of the bargain, creates an express warranty that the goods shall conform to the affirmation or promise.
    - b. Any description of the goods which is made part of the basis of the bargain creates an express warranty that the goods shall conform to the description.
    - c. Any sample or model which is made part of the basis of the bargain creates an express warranty that the whole of the goods shall conform to the sample or model.
2. It is not necessary for the creation of the express warranty that the seller use formal words such as "warrant" or "guarantee" or that he have a specific intention to make a warranty, but an affirmation merely of the value of the goods or a statement, purporting to be merely the seller's opinion or commendation of the goods does not create a warranty.

As is always the case in law, what seems simple and obvious language is subject to the many variations which arise in the empirical world. Thus my client's statement about what he should get for "perfect money" is subject to whether the parties were dealing in new, off-grade, or used goods. The excerpt of the UCC is also subject to those questions and many more such as the use of "equal" components, the misuse of products, and trade usages of terms.

It is ironic that when you ask for a simple and just requirement from a manufacturer—that you receive what you bargained for—an avalanche of laws and statutes descends upon you, leaving an entanglement that can become excruciating once you try to collect. At this stage of the warranty evolution your best defense against being stuck with a malfunctioning computer and its concomitant problems is to check the machine out fully before you take it home. ▼

Solution to last month's PROMpuzzle

■	■	M	A	R	■	L	O	R	C	A	■	T	I	S	■	■
■	P	A	G	E	■	A	N	T	O	N	■	A	R	U	G	■
K	O	R	E	A	■	R	E	E	N	T	E	R	A	B	L	E
M	L	I	■	D	I	G	S	■	C	R	T	■	S	C	A	N
C	O	N	S	O	L	E	■	I	E	E	E	S	■	O	D	D
■	■	■	H	U	E	■	E	S	P	S	■	E	S	M	E	S
O	C	T	E	T	■	S	C	O	T	■	I	T	E	M	■	■
W	A	R	D	■	H	A	R	P	■	S	C	A	L	A	R	S
E	R	A	■	S	I	M	U	L	A	T	E	S	■	N	E	A
S	E	N	S	O	R	S	■	A	R	E	S	■	R	D	T	S
■	■	S	O	L	E	■	T	N	T	S	■	F	U	S	E	S
R	A	D	I	O	■	S	H	A	S	■	P	O	I	■	■	■
E	D	U	■	S	H	E	E	R	■	T	U	R	N	O	F	F
A	O	C	S	■	I	V	E	■	A	I	N	T	■	R	O	E
P	R	E	I	N	D	E	X	I	N	G	■	R	O	B	O	T
■	E	R	T	E	■	R	I	G	O	R	■	A	D	I	T	■
■	■	S	E	W	■	S	T	O	N	E	■	N	E	T	■	■

PROMqueries

HAVE A PROBLEM? ASK ROM

by Eben Ostby

Dear ROM:

What is structured programming? I've seen it mentioned frequently enough to wonder if it's just a buzz word or if there's more to it.

Corinne Cannon
Kingston, Pennsylvania

Dear Corinne,

"Structured programming" is a phrase that has been tossed around a lot these days. Based on ideas put forth by E. W. Dijkstra, among others, structured programming is a highly disciplined programming method. Some of the things involved are:

Programming without GOTOs. *If you use BASIC, you probably can't live without the GOTO statement. But in other languages it's somewhat easier without than with. A structured program will be written as blocks of code that are executed by IF statements and FOR or DO statements. Rather than branch around parts of a program with a GOTO statement, a structured program will treat that part of the program as a block of statements, and either execute the entire block or not, depending on the conditional (IF) statement.*

A modular language. *Many languages, following in the footsteps of ALGOL, are highly "modular." Typically, these languages stress the use of subprograms or functions. Also common is a facility that allows a series of statements to be substituted for a single statement anywhere one is legal. In this way, a program can be defined to be any single statement, a definition that keeps things simple. Then, to extend the meaning of a program, you can just substitute a number of statements in for the single one. On paper, this appears to be a useless distinction, but it does result in a modularity where substitutions to programs can be made easily.*

A "top-down" approach. *Programming is easier if the program is defined in very general terms first. Then each definition is successively refined. When it can be refined no more, the result is a detailed description of the program as a series of subprograms, each of which call subprograms. (I give an example of this in my article "Chart UP and Flow Right" in the October issue of* ROM.)

The intent in structured programming is to produce more reliable programs. It seems to work.

ROM

Dear ROM,

Some friends were discussing the halting problem paradox. I tried to get them to halt long enough to ask them what on earth they were talking about, but I didn't succeed. Exactly what is the halting problem?

John Tentler
Red Bank, New Jersey

Dear John,

The halting problem is a fascinating proof based on the theory of automata, which are elementary models of computers. In particular, the paradox rests on the theory of Turing machines, which are sophisticated (yet simple) automata devised by the English mathematician Alan Turing. Joseph Weizenbaum provided an eloquent description of them in "Computer Power and Where It Comes From" in the August issue of ROM, *so I won't attempt one here.*

Basically, what the halting problem says is that there is no way to construct a Turing machine (and hence a computer) that will determine if another machine will ever stop executing. Put another way, you can't write a program to detect whether another program (suppose it is given as input to the first program) will loop forever or stop looping some day.

Marvin Minsky's proof of the fact is wonderfully simple. He constructs a machine that will stop running if the program it receives as input will stop too; otherwise it will run forever. Then he constructs another machine from the first one, one that will stop only if the first one would have run forever, and vice versa. Finally, he feeds a description of the second one to itself—so it will stop only if it will run forever, and it will run forever only if it will stop. Since this is patently absurd, the assumption that the first machine was possible to make must also have been absurd. This proves that such a machine cannot be made.

ROM

Dear ROM:

I've come across a number of references to the Gray code. Could you tell me who or what it exactly is?

Thomas Barlow
Milwaukee, Wisconsin

Dear Thomas,

The Gray code is something you're not likely to run into in home computing much, and if you did, it would be because you had a machine tool or a wind-vane attached to your machine. The code, patented by F. Gray, is a method of encoding decimal numbers as binary integers.

The standard method of encoding decimal as binary, which is known simply as BCD (binary-coded-decimal), expresses a single decimal digit as a four-bit binary number. If you're familiar with binary numbers, you know that this means that a decimal two is coded as 0010, three as 0011, four as 0100, and so on.

Now, suppose you have some device that rotates and needs to send information about the position of its rotating parts to your computer—a wind-vane is a good example. To encode the information for transmission to the computer, you might have a disk attached to the rotating part, and this disk would have a number of ring-shaped regions divided into segments on it.

A sensing device, like a switch or a feeler, could contact the disk, and its state would then depend on the position of the disk—or more properly, the region just below the switch. Since the switches can be either on or off, it would be simple to arrange the regions on the disk so that the switches "counted off" the position of the rotating shaft in, say, BCD (one bit per switch). All the computer has to do is sample the state of the switches and it instantly knows the wind direction (for instance).

But a problem arises when the thing turns. Since wind doesn't necessarily turn in exact, one-sixteenth of a turn increments, it would be possible for the thing to turn between *sensing positions. Suppose, for instance, the shaft turned to position number three, sending a binary 0011 to the switches. Then suppose it turned just a bit farther. Now suppose the computer decided to sample the switches. As the thing turns, the switches change from one state to the next.*

In this case, three of the four switches change—two from "on" to "off," and one from "off" to "on." If the switches are in the process of switching just as the computer samples them, some may have changed states while others haven't quite. So the computer may think that the device was in state five (0101) when in truth it was in state four (0100).

What the Gray code does is rearrange the sequence of binary numbers so that only one bit changes at a time—that is, in changing from state three to state four, one binary bit (or switch) changes instead of three. So the sequence for a four-bit Gray code looks like this: 0000 0001 1001 1101 0101 0111 1111 1011 0011 0010 1010 1110 0110 0100 1100 1000.

Are you planning a computerized weather station?

ROM

Dear ROM:

What exactly is an interrupt, and how does it function?

Wayne Harper
Decatur, Arkansas

Dear Wayne,

When a computer has to communicate with its peripherals—printers, disk drives, keyboards, etc.—the computer and the devices are rarely ready at the same time. For instance, if you could type as fast as the computer could accept input, you'd have the world's fastest secretary beat a millionfold (give or take an order of magnitude).

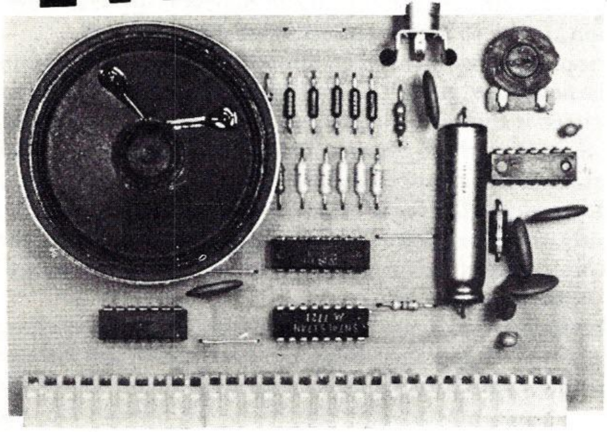
The system many computers use to get around this is to have the peripheral device interrupt when it's ready for the computer to send it information, or when it is ready to send information to the computer. The interrupt causes the processor to stop what it's doing and go to another program which simply interacts with the peripheral. Then it goes back and does what it was doing previously. In this way a computer can also "save time" by doing computations while it's waiting for you to type something in, or for a printer to type a long report out.

ROM

NEWTECH
Model **68**

PRODUCES MELODIES, RHYTHMS, SOUND EFFECTS, MORSE CODE, TOUCH-TONE SYNTHESIS, AND MORE!

MUSIC BOARD



- Compatible with Southwest Technical Products Corp. 6800 computer system. Uses one I/O slot. • Glass epoxy printed circuit board with high quality components. • 6-bit latching digital-to-analog converter. • Audio amplifier • Speaker • Volume control • RCA phono jack for connection to external speaker or audio system.
- 60-day parts and labor warranty

Complete user's manual with BASIC program for writing musical scores and 6800 Assembly Language routine to play them. SWTPC AC-30 compatible cassette with above programs.

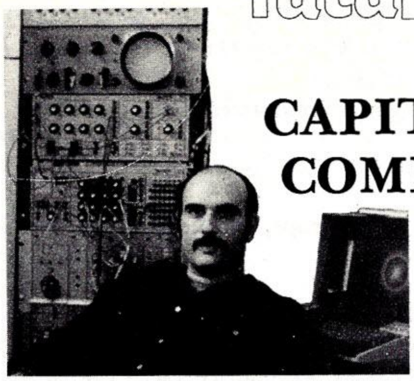
$59.95 AVAILABLE THROUGH YOUR LOCAL COMPUTER STORE

ASSEMBLED AND TESTED

NEWTECH COMPUTER SYSTEMS, INC.

131 JORALEMON STREET * BROOKLYN, NEW YORK 11201 * (212) 625-6220

futuROMa



THE CAPITALIST COMPUTER

by Bill Etra

Here's what happened. It was really totally accidental. Hollerith, followed by International Business Machines, brought out digital systems for processing information, and then binary digital systems for the same purpose. These machines meshed perfectly with the American free-enterprise capitalism. Very shortly, digital information systems in the guise of business machines were in wide use throughout the country.

During World War II, when these machines first came into existence, the people involved couldn't really figure out any civilian use for computers, or even how large they should be. Computer research might well have ceased with armistice, except for the fact that businesses began to see the possibilities of implementing their already existing business practices on the new machines.

These people started to pick up computer systems for accounting purposes. From there on, a sort of geometric progression began—the more accounting you do on the machines, the more you need the machines to do accounting. Take credit cards, for instance. They are a total creation of computers. You couldn't economically keep credit card records any other way. Once computers created credit cards, they became indispensable to the modern banking community. The way our banks now work actually makes them totally dependent on computers.

This dependence has even begun to affect the legal framework within which we operate; a rash of public action suits have appeared against banks for overdrafts and for delaying funds. With computers it doesn't take more than a few hours to transfer funds all over the country. And even though banks have been using this method for years, the public has not had the benefits passed through to it. In any case, from an operational point of view, modern banking can't be done without computers. And so the market in this area has exploded.

This explosion permitted computer scientists to experiment as did the funds generated by the military complex's need for specialized machines. Even these military developments were advantageous for the computer community. If you made a good machine for the military, you could also sell it to business. They needed better business machines and, as far as the computer was concerned, it was basically just a difference of input: tactical data or dollar data. The designers weren't developing one-of-a-kind machines—you could downgrade or upgrade the computer and make a highly viable and commercial civilian product out of it.

Enter the space program. It poured a lot of money into research to reduce the computer's physical size while at the same time increasing its computational capacity. As with the war effort, almost everything from the space program could be absorbed by our industry because of the way the capitalistic system works.

The need for data processing became tremendous at this point. We had a large middle-income group of people, all of whom had money to spend. But spending it in traditional ways became cumbersome. Which is why mail-order houses and credit-card companies began their spectacular growth, all dependent on the newly developed computers.

By contrast, the Russians didn't have any of this. Of course the Russians had scientists who could build a large computer, and they did—for military use. But they couldn't build a hundred of them or two hundred or two thousand. There simply was no market to support such demand. Yet it's when you build two thousand that you really improve the machine. When you're in a competitive market somebody else looks at the machine and says, "Yech" and builds a better one. The Russians were only building them one here to operate a plant, or one there to sample statistics, which is the main reason they were left behind in the computer revolution. Microelectronics is, to a large extent, a product of the data-processing boom. Because of the capitalistic consumption of computers we have thousands of engineers, hundreds of thousands of people employed in data processing, thousands and thousands of programmers, thousands of companies manufacturing devices. In order to catch up to us in microelectronics, a country would need hundreds of these specially trained people.

Microelectronic development isn't one simple stage. There are specialized applications for designers, for microelectronic photographers, and so on. These specialties have developed into entire industries as subsets of the boom we underwent due to the demands of capital transfer.

What is going to happen in the future, particularly with the Japanese, is more difficult to ascertain. They are sinking their money into bypassing large-scale integration entirely.

Now IBM already has this next generation of systems. One where you directly record the entire layout of the chip with an electron beam on the material. You can then make customized chips. You've put in the x-y computer data on the set of screens, put it into the memory and *it* writes a new device to do what you want. Presently this takes a long time for IBM. But that's certainly the next level. And that's where the Japanese are trying to catch up or even leapfrog. Matsushita, etc. are attempting to bypass large-scale integration, going to direct recording of chips instead.

The problem is you need computers to check the diagrams for the new chips. It can't be done by hand any more. Nobody can check it that way. So, although the Russians might be able to go directly to electron beam recording for chips or some even more advanced technology, when it comes to mass production they'll face the original problem. As an example, recently when they took a Russian plane apart, they found the electronics composed primarily of vacuum tubes. Very small, very neat, but still vacuum tubes.

This seems to indicate that they are not going to catch up very easily at the rate they are going, particularly not without importing capitalistic computers to help them. Currently they can't manufacture the electronics for missiles like the Cruise, and that's why they can't sign the Strategic Arms Limitation agreement. We have weapons that can out-think them. They still don't have the massive technological demand for data processing which the consumer society has produced. The demand that literally allowed us to spend ourselves into computer innovation. ▼

Get updated . . . keep updated with

BYTE

the leading magazine in the personal computer field

The personal computer age is here.

Join Byte's 110,000 subscribers and catch up on the latest developments in the fast-growing field of microprocessors. Read BYTE, The Small Systems Journal that tells you everything you want to know about personal computers, including how to construct and program your own computer (over 30,000 BYTE readers have already built, or bought, their own systems and half of these have 8K bytes or more).

You'll find our tutorials on hardware and software invaluable reading, also our reports on home applications and evaluative reviews based on experiences with home computer products.

Home computers . . . practical, affordable.

Large scale integration has slashed prices of central processors and other computer components. This has encouraged the development of new, low-cost peripherals resulting in more hardware and software — more applications than you could imagine, more opportunities for you. BYTE brings it all to you. Every issue is packed with stimulating and timely articles by professionals, computer scientists and serious amateurs.

BYTE editorials explore the fun of using and applying computers toward personally interesting problems such as electronic music, video games and control of systems for alarms to private information systems.

Subscribe now to BYTE . . . The Small Systems Journal

Read your first copy of BYTE, if it's everything you expected, honor our invoice. If it isn't, just write "CANCEL" across the invoice and mail it back. You won't be billed and the first issue is yours.

Allow 6 to 8 weeks for Processing.

© Byte Publications, Inc. 1977

BYTE Subscription Dept. 85 • P.O. Box 361 • Arlington, Mass. 02174

BYTE

PLEASE ENTER MY SUBSCRIPTION FOR:

☐ One year $12 (12 issues) ☐ Two years $22 ☐ Three years $32

☐ Check enclosed (entitles you to bonus of one extra issue)

☐ Bill me ☐ Bill BankAmericard/Visa ☐ Bill Master Charge

Card Number: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date: ____________

Signature: ____________ Name *(please print)* ____________

Address: ____________

City: ____________ State/Country: ____________ Code: ____________

FOREIGN RATES FOR ONE YEAR: *(Please remit in U.S. Funds)*

☐ Canada or Mexico $17.50 ☐ Europe $25 (Air delivered)

☐ All other countries except above: $25 *(Surface delivery)*

Air delivery available on request

COMPUTERS AND CRYPTOGRAPHY

by Frederick W. Chesson

The average computer hobbyist, and even the professional user, may not suspect the close kinship between cryptology, the art and science of secret writing, and the entire realm of data processing. Each has done much to nurture the growth of the other. In the early 1930s, IBM card tabulators were first used by the United States Navy and then the Army for cryptanalysis or code-breaking. Today microcomputers have revolutionized data handling and made mandatory the securing of vital information and electronic fund transferral by highly sophisticated cipher systems.

Just as data processing has its own lexicon and terminology—words like *online, RAM, ROM,* and *PROM*—so does cryptology. *Cryptography* concerns the written word—how letters, figures, and words may be transformed to render them inscrutable to the uninitiated. *Enciphering* is the process by which this transformation takes place. *Deciphering* is the reverse process, where the data is again made understandable, usually to the legitimate recipients of the message. The enciphered message is called a *cryptogram.* An attempt to break the enciphering process is variously called *decrypting, cryptanalysis,* or just plain old-fashioned *code-breaking.*

Code and *cipher* are sometimes used interchangeably in describing secrecy systems, but *code* is, more strictly speaking, a process by which a complete word, a phrase, or even one or more entire sentences are replaced by a group of letters or numbers (rarely mixed). Before the advent of the teletypewriter and satellite communication systems, commercial cable codes carried the world's commerce in terse, sometimes secret, phrases.

In the New Standard Code of 1929, for instance, the phrase "You are authorized to advance ship's captain for disbursements" becomes AQOIV, thereby saving some eight words of often costly cable tariffs. The numeric group 513229 might accomplish the same thing in another commercial code book.

Both alphabetical and numerical systems could be enciphered for added secrecy, though the majority of commercial books were oriented toward convenience and economy. Large industries like shipping and international banking required the compilation of highly specialized code books of several volumes totaling up to 300,000 entries.

Ciphers, on the other hand, generally operate on each individual letter, or sometimes on pairs or trios of letters. This covers a vast field in which there are two major subdivisions: *transposition* and *substitution.*

Transposition keeps the original letters (or even words) intact, but routes them into new alignments via set routes. The word CODE can be scrambled into such diverse forms as DOCE, EDCO, ODEC, and so on. During the American Civil War the Union's U.S. Military Telegraph Corps relied exclusively upon word transposition systems, with an increasingly generous usage of code words. In one code book, used in 1862, BREMEN at the start of a message meant a transposition block of seven lines and six columns, with the text written in by going up the fourth column, down the third, up the fifth, down the second, up the sixth, and down the first. When code words for important persons, places, and military terminology were added (Lincoln = Berlin, Richmond = Humming, Attacking = Oystering), the resultant system was enough to baffle Confederate wire-tappers.

The South, for its cryptosystems, relied upon a cipher which was already some three hundred years old. This was the Vigenere, a classic substitution cipher, in which every letter was replaced, according to a key word, by some other letter. While in theory a very secure system for its time, the employment of the Vigenere (or Court Cipher as it was called because of its association with the French Royal Court of Louis XIV) was often faulty and subject to being broken by the Yankees. Perhaps the two gravest errors were the long-continued usage of a particular code word (MANCHESTER BLUFF, a suburb of Richmond, was a protracted favorite) and only partial encipherment of messages. The latter fault gave the code-breakers many leads to guess at in order to recover the code word for a whole series of past and future messages.

The ancestor of the Vigenere substitution cipher, with its easily variable code word, was the monophonic, or simple substitution cipher, a form still present today in the recreational cryptograms often found in the crossword puzzle section of many newspapers.

Monophonic ciphers were around as early as the twelfth century or earlier in Arabic lands and soon found their way to prerenaissance Italy, where they were adopted by the host of squabbling city states which crowded that tortured peninsula. By the sixteenth century, Venice, Florence, the Vatican, and other ministates had found it necessary to improve upon simple substitution by giving common letters—such as E, A, T, I, and O—a variety of substitutes, usually exotic symbols. As more and more symbols and letter and number combinations were added, the *nomenclator* was born. The phrase "Your Gracious Majesty" could now be represented by &J, saving considerable time for the harried cipher clerks. The eventual evolution of the nomenclator was the familiar military and diplomatic code book, whose security was quite literally a matter of life and death.

Ultimately, the code book went electronic, disappearing into a disk file, core memory, or ROM, so that the pattern 101001101100 may be read out, for instance, as "Your order will be processed as soon as our programmer returns from vacation, please be patient."

Our Civil War was responsible for an explosive growth of American inventiveness, as evidenced by a flood of new patents. Cipher apparatus was no exception, as witness patent numbers 48,681 and 50,946, which were granted for disk and leaf type arrangements for producing what was essentially the Confederacy's borrowed Vigenere system.

In succeeding decades these initial drops in the bucket were followed by a growing flood in which America had no monopoly, as witnessed by the Swedish Patent No. 4878 of 1893. On it, an appropriate Old Norse Dragon guards a hand-operated version of the old cipher-disk, in which the alphabet may be well mixed by removing and relocating the little letter-marked inserts. With a long and random key word, this system could offer considerable security to the user.

As early as the Gay Nineties, Dr. Hollerith's punch cards and their electromechanical readers, designed to assist the census of 1890, could have done frequency counts and other cryptanalytic statistics.

It remained for World War II to fully initiate the electrical-electronic explosion of cryptography. The dozen or so IBM tabulators in use by the military at the time of Pearl Harbor had perhaps multiplied by a hundredfold at the time of V-J Day in 1945.

During the war, highly specialized calculators, which approached true computer status in their internally programmed structure, were in use, especially in America and Britain. In the United States, the Japanese diplomatic cipher machine, known to us as "Purple," was cracked long before the Pearl Harbor attack, the plans for which were kept a tight secret from the diplomats by the imperial staff. In England, the still largely secret Colossi machines routinely broke Germany's widely-employed Enigma machine, enabling the Allies to anticipate most of Hitler's secret battle plans, even before his own generals were fully informed.

To forestall future atomic Pearl Harbors, this country now relies upon the Central Intelligence Agency to gather and interpret information affecting the nation's well-being. Protecting our own communications and breaking the codes of potential enemies is the duty of the National Security Agency, headquartered in Fort Meade, Maryland. The N. S. A., in its continuing search for better and faster data-processing equipment, has done much to spur the growth of microelectronics. Its achievements are still largely undisclosed for security reasons: even the executive order establishing it in 1952 is still said to be under wraps. The initials of the organization define its shadowy public image: "N. S. A. = Never Say Anything"!

Cryptographers have met with outstanding successes and failures over the years in deciphering cryptograms of historical interest. Champollion's cracking of the hitherto unbreakable heiroglyphic "alphabet" of ancient Egypt in 1821 employed cryptanalytic techniques, as did Michael Ventris in his solutions of the Cretan Linear-B script in 1952.

Other ancient languages, such as the Mayan pictographs, are still largely unsolved. One of the most interesting of documents is the famous Voynitch Manuscript, now in Yale University's Beinecke Rare Books Library. Supposedly attributed to the English prerenaissance philosopher Roger Bacon, the work, filled with vividly colored botanical and astrological fantasies, may actually have been produced in the early seventeenth century from much earlier material in order to bilk Emperor Rudolph II out of literally a "princely sum." Only now have its ciphers within ciphers begun to be unraveled.

In more modern times, the Beale Papers have become a widely-sought-after source of a treasure of gold, silver, and gems, all deposited circa 1820 in the Blue Ridge Mountains of Virginia, near the famed Peaks of Otter. Of the three papers, each containing a separate message, Number Two has been deciphered, using the Declaration of Independence as a key. This gave a general description of the treasure and its location. But Papers Numbers One and Three, supposedly giving exact instructions to the trove and a list of the treasure party's next of kin, have so far eluded computers, ESP practitioners, and simple pick-and-shovel wielders alike.

Far more sinister are the Zodiac Messages, penned in 1959-60 by "Zodiac", in the San Fransico Bay area. The alleged killer of possibly a dozen persons tantalyzed public and police alike by sending a series of cryptograms to various newspapers. Only one message was known to be solved; in it the writer boasted of his desire for killing victims who would then become his "slaves." It is possible that the other texts were carefully contrived hoaxes, so structured as to appear to bear authentic information.

Most interestingly, both the deciphered Beale and Zodiac cryptograms were of the same cipher type, the Homophonic, in which the letters of the alphabet were given a variety of substitutes. In the Zodiac message, the letter E had seven substitute symbols and A had five; while in the second Beale paper, T had seventeen substitutes and E fourteen.

Although highly sophisticated programs have been developed for the analysis of the frequency, contact, and other statistical qualities of these two now famous cipher systems, it does not take much in the way of hardware or software to make *trial substitutions* in a search for possible solutions. Even so primative a device as a basic TV typewriter may be easily modified to provide character substitutions at the touch of the keyboard. This possibility, along with other aspects of cryptography, will be discussed in another issue. ▼

Further Reading

Kahn, David. *The Code Breakers.* New York: Macmillan Co., 1967. This is probably the most definitive modern work on cryptology, covering both historical and technical aspects. See (in the hardcover edition) chapters 2 and 3 for the origins of modern cryptography, chapter 7 for Civil War ciphers, and chapter 19 for the National Security Agency. Pages 1101-2 in the notes section discuss cryptographic patents.

Gaines, Helen F. *Cryptanalysis.* New York: Dover Publications, 1956. A paperback classic covering all aspects of modern cipher systems and their analysis.

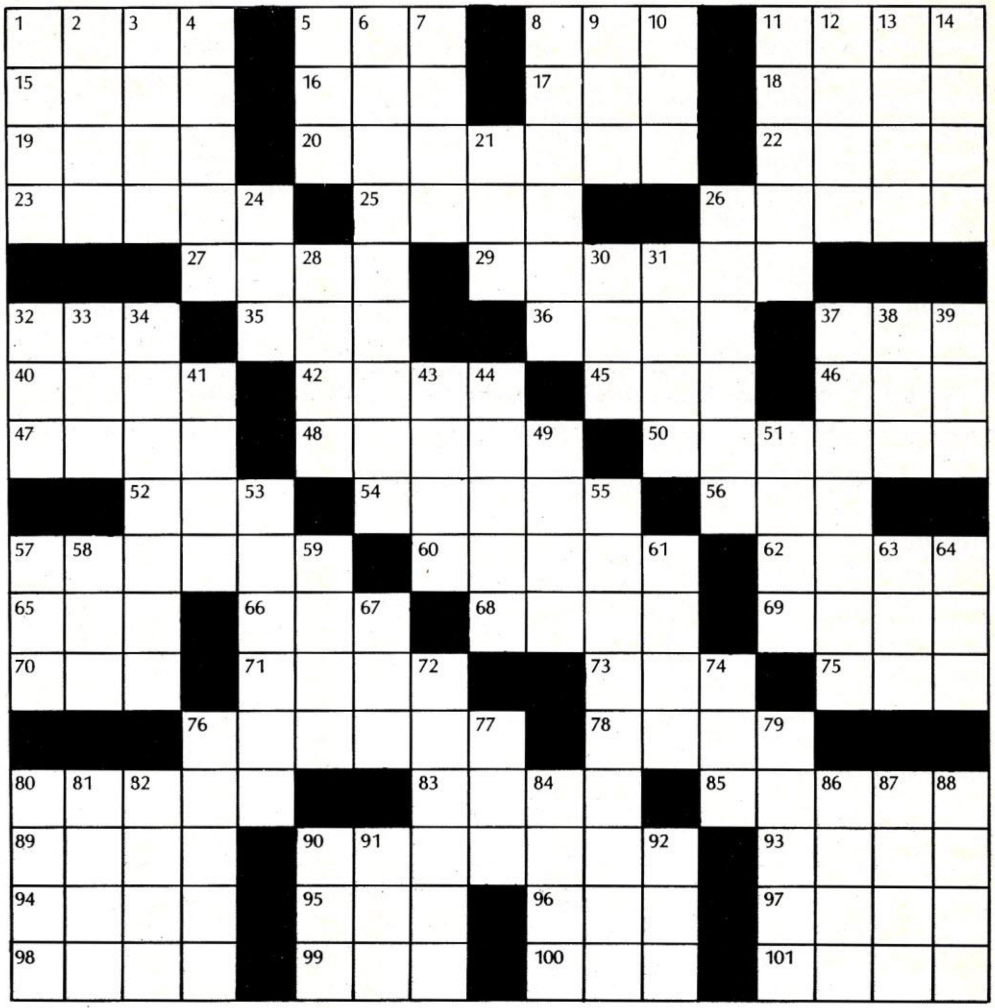
Winterbotham, W. F. *The Ultra Secret.* New York: Harper and Row, 1974.

Brown, Anthony Cave. *Bodyguard of Lies.* New York: Harper and Row, 1975. These two books lift the curtain of thirty years of top secrecy, revealing how Germany's main cipher machine system was cracked by the first true computers and the military consequences thereof.

Cryptographic patents may be found occasionally in the weekly "Official Gazette" of the U.S. Patent Office, generally under the following classifications in the electrical section of the journal: 178-5, 178-22, 179-1, and 197-4. Individual patents may be obtained for fifty cents each from The Commissioner of Patents, Washington, D. C. 20231. Since secrecy resides in the keying system, generally speaking the publication of a modern secrecy system does not necessarily compromise its security. . . though there can be exceptions!

PROMpuzzle

by Daniel Alber


1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18
19 20 21 22
23 24 25 26
27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46
47 48 49 50 51
52 53 54 55 56
57 58 59 60 61 62 63 64
65 66 67 68 69
70 71 72 73 74 75
76 77 78 79
80 81 82 83 84 85 86 87 88
89 90 91 92 93
94 95 96 97
98 99 100 101

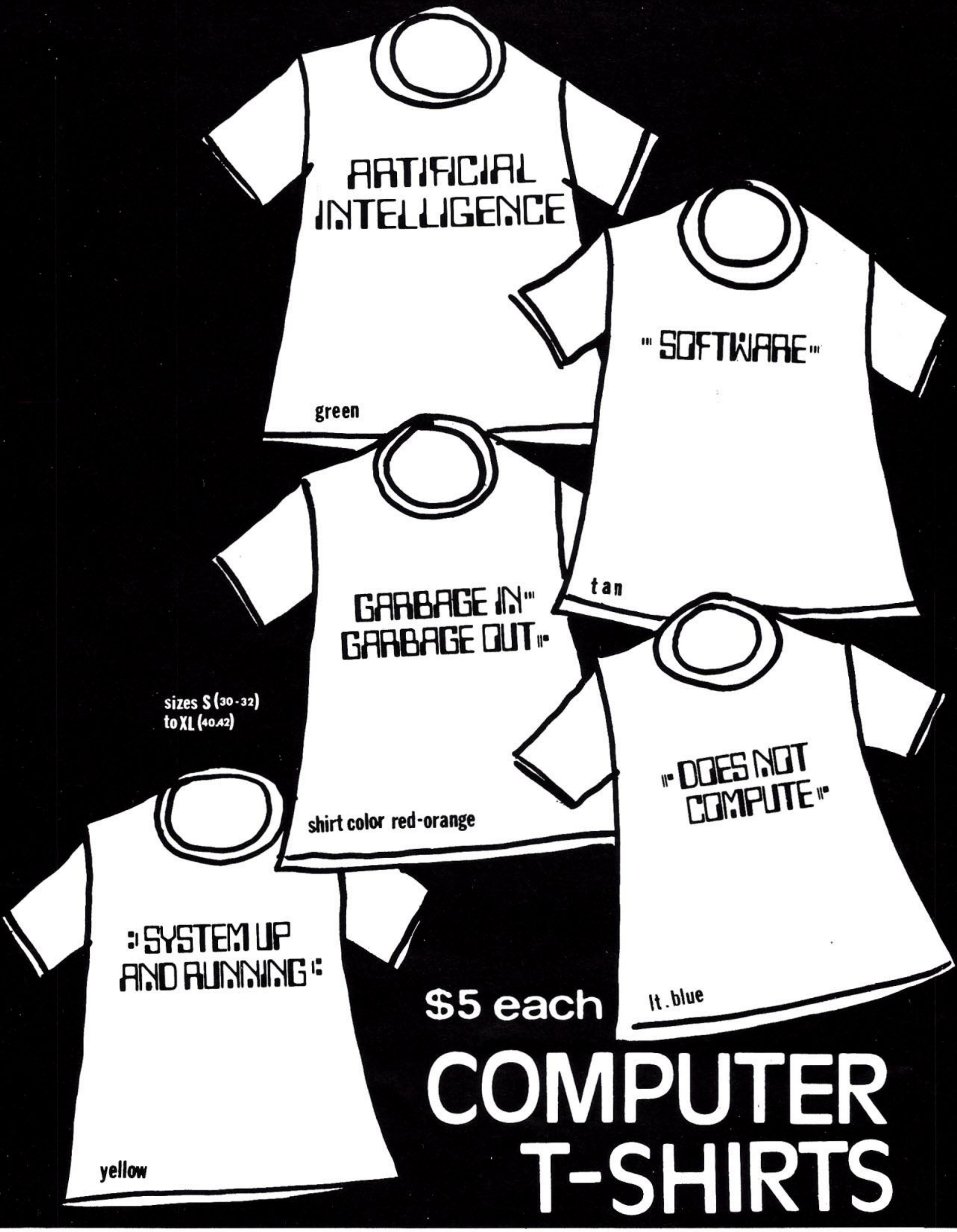
The solution to this PROMpuzzle will appear in next month's *ROM*.

ACROSS

1. Complementary metal-oxide-semiconductor (abbr.)
5. Auxiliary power plant (abbr.)
8. Spigot
11. Search for data
15. Past tense of heave
16. Ghostly cry
17. ____ Baba
18. Top notch
19. Notion (Fr.)
20. ____ circuit
22. Color
23. Makes coffee
25. Information system access lines (abbr.)
26. Judged
27. A group
29. Data placement at a specific address
32. Drunkard
35. Hawaiian garland
36. Strike
37. Type of memory
40. Section
42. Birds (Lat. pl.)
45. Electrical unit
46. Part of the psyche
47. "Dove Sono" for example
48. Jumped
50. Subordinate computer equipment
52. Sash (Jap.)
54. Retain data
56. Compass reading
57. Produce an electrical charge
60. Spirits
62. ____ structures
65. Born
66. Data links (abbr.)
68. Kin of the onion
69. This (Sp.)
70. Two in Spain
71. Islands (Fr.)
73. Government agency (abbr.)
75. Lend an ____
76. Between characters
78. Teleprinters (abbr.)
80. High amplitudes
83. Close by
85. Type of Read Only Memory
89. Hindu queen
90. Programming instructions
93. Shakespearean villian
94. English river
95. Prefix with edit or assembly
96. Marriage vow (2 wds.)
97. Was informed
98. Longings
99. Took a chair
100. Morning moisture
101. Summers in Lyons

DOWN

1. Microcomputer component
2. Binary ____
3. ____ and out
4. Searches
5. Goat hair
6. Some charges
7. Cooking vessels
8. Ordered data
9. "____ in the Family"
10. Quiche
11. The devil
12. Crazy person (sl.)
13. Girl's name
14. Want
21. Swiss river
24. Simulation oriented language (abbr.)
26. Voltage ratio
28. ____ time
30. ____ mode (2 wds.)
31. Water barriers
32. Resort
33. Paddle
34. Three-electrode devices
37. ____ bias
38. Mature
39. Metal-oxide semiconductor (abbr.)
41. Forbidden
43. Consumes
44. Thread holder
49. Genuine
51. Penny ____
53. Some packages
55. Conducting element
57. Hoosier State (abbr.)
58. Prefix for new
59. Miss Fitzgerald
61. Blank instruction
63. Greek letter
64. Poetic contraction
67. Triple ____
72. Trumpet call
74. Exist
76. Outer coverings
77. Place a storage device
79. Nail
80. Implore
81. Roof part
82. Shortly (archaic)
84. Dry
86. Carry on
87. Curved molding
88. Cuts the grass
90. Central processing system (abbr.)
91. ____ *pro nobis*
92. ____ speed storage


ARTIFICIAL INTELLIGENCE
green
"SOFTWARE"
tan
"GARBAGE IN" "GARBAGE OUT"
shirt color red-orange
sizes S (30-32) to XL (40-42)
"DOES NOT COMPUTE"
lt. blue
:SYSTEM UP AND RUNNING:
yellow

$5 each

COMPUTER T-SHIRTS

name ____________________

address ____________________

city ________ state ____ zip ______

mail to MARTHA HERMAN 114 W. 17 STREET
NEW YORK 10011
(212) 691-2821

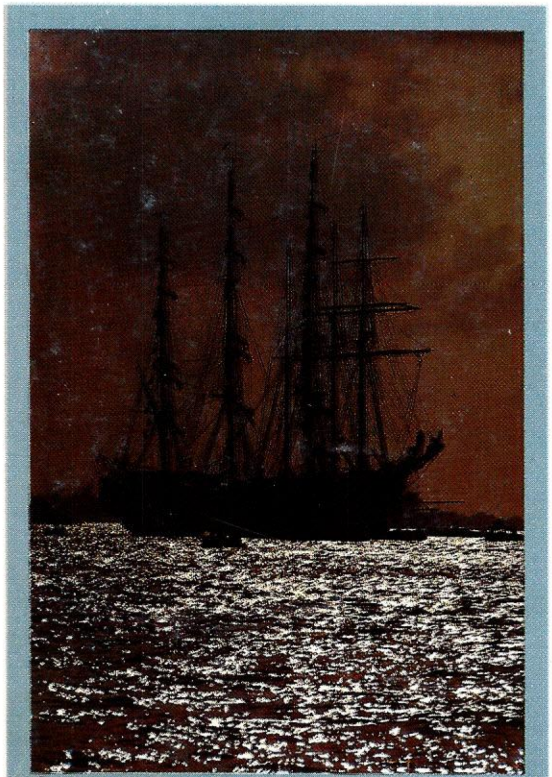
quantity	chest size	design	$	
add .60 per shirt for shipping				
		total enclosed		

A magnificent first edition . . . pictorial memories of a moment America will never forget . . . or see again

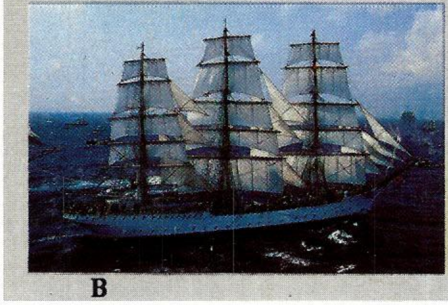
THE SAILING SHIPS

Professional quality lithographs of the majestic Bicentennial Ships, now available to the public in an exclusive, limited edition from America's foremost engraver-printer, COLLIER GRAPHICS.

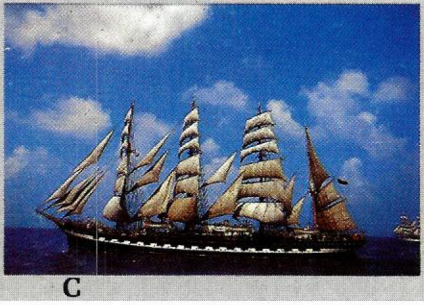
handsomely matted in silvery metal frames; $25 each **unframed, $10 each**



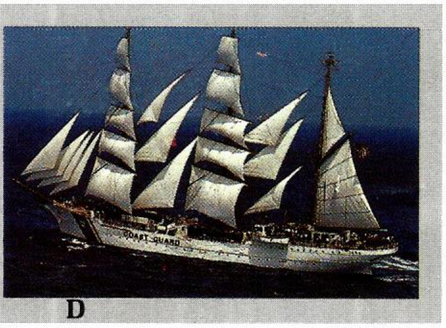
A



B



C



D

THIS IS A LIMITED EDITION . . . ORDER TODAY WHILE THEY LAST

Never before offered to the public, these magnificent 12x19 full color lithographs of the tall-masted SAILING SHIPS were originally photographed for professional use only!

Now, you can own and enjoy their historic beauty, their incredible clarity, color and reproduction, which gives them almost a three-dimensional quality. And they are only available from COLLIER GRAPHICS — who provide the superb color graphics for America's leading advertising agencies and publishers.

Perfect for your home, boat or office. A lasting gift. Order a set for yourself and one for your children . . . to keep always.

COLLIER GRAPHICS INC., 240 West 40th Street, New York, New York 10018

Please rush the following SAILING SHIPS, securely packaged and postage prepaid, with the understanding that my purchase is UNCONDITIONALLY GUARANTEED by you if the order is returned within 15 days of delivery:

Send me the following print(s). Quantity ________ Price ________ Total ________

A. Christian Radich ________

B. Danmark ________

C. Kruzenshtern ________

D. Eagle ________

Please add $2.50 for shipping and handling for framed print(s) or $1.00 for unframed print(s).
(N.Y. State residents add applicable tax, NYC residents add 8%. Allow 6 weeks for delivery.)

Name ________

Address ________

City ________ State ________ Zip ________

Enclosed, check or money order for $________

Or charge my credit card ________ Master Charge ________ BankAmericard

________ Credit Card # ________ Inter Bank # ________ Expiration Date

Signature X ________